7-2017

Symbolic and Visual Retrieval of Mathematical Notation using Formula Graph Symbol Pair Matching and Structural Alignment

Kenny Davila Castellanos
kxd7282@rit.edu

Symbolic and Visual Retrieval of Mathematical Notation using Formula Graph Symbol Pair Matching and Structural Alignment

by

Kenny Davila Castellanos

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Computing and Information Sciences

B. Thomas Golisano College of Computing and
Information Sciences
Rochester Institute of Technology

July 2017

Signature of the Author ____

Certified by ____
PhD Program Director Date

Ph.D. IN COMPUTING AND INFORMATION SCIENCES
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

Ph.D. DEGREE DISSERTATION

The Ph.D. degree dissertation of Kenny Davila Castellanos
has been examined and approved by the
dissertation committee as satisfactory for the
dissertation required for the
Ph.D. degree in Computing and Information Sciences

Dr. Richard Zanibbi, Dissertation Co-Advisor

Dr. Stephanie Ludi, Dissertation Co-Advisor

Dr. Daniel Phillips, External Chair

Dr. Linwei Wang

Dr. Cecilia Ovesdotter Alm

Dr. Christopher Kanan

Date

Symbolic and Visual Retrieval of Mathematical Notation using Formula Graph Symbol Pair Matching and Structural Alignment

by

Kenny Davila Castellanos

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences Ph.D. Program in Computing and Information Sciences
in partial fulfillment of the requirements for the
Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

Large data collections containing millions of math formulae in different formats are available online. Retrieving math expressions from these collections is challenging. We propose a framework for retrieval of mathematical notation using symbol pairs extracted from visual and semantic representations of mathematical expressions on the symbolic domain for retrieval of text documents. We further adapt our model for retrieval of mathematical notation on images and lecture videos. Graph-based representations are used on each modality to describe math formulas. For symbolic formula retrieval, where the structure is known we use symbol layout trees and operator trees. For image-based formula retrieval, since the structure is unknown we use a more general Line of Sight graph representation. Paths of these graphs define symbol pairs tuples that are used as the entries for our inverted index of mathematical notation. Our retrieval framework uses a three-stage approach with a fast selection of candidates as the first layer, a more detailed matching algorithm with similarity metric computation in the second stage, and finally when relevance assessments are available, we use an optional third layer with linear regression for estimation of relevance using multiple similarity scores for final re-ranking. Our model has been evaluated using large collections of documents, and preliminary results are presented for videos and cross-modal search. The proposed framework can be adapted for other domains like chemistry or technical diagrams where two visually similar elements from a collection are usually related to each other.

Co-authorship

A large portion of Chapter 3 was written in collaboration with my advisor Richard Zanibbi, and also Frank W. Tompa and Andrew Kane from University of Waterloo. and it has appeared before in International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR 2016 and 2017), and also in the proceedings of the 12th NTCIR Conference on Evaluation of Information Access Technologies. Chapter 4 was also written in collaboration with my advisor Richard Zanibbi, and has been accepted for publication in proceedings of the 14th International Conference on Document Analysis and Recognition (ICDAR 2017). Finally, Chapter 5 was written in collaboration with my advisors Richard Zanibbi and Stephanie Ludi, and has appeared before in the International Conference on Frontiers in Handwritting Recognition (ICFHR 2014).

Acknowledgments

Dr. Richard Zanibbi and Dr. Stephanie Ludi for their constant support. Dr. Frank W. Tompa and Dr. Andrew Kane for their contributions to the Tangent-3 system, including the original core engine implementation that incorporates iterators trees for fast list merging developed by Andrew Kane. Dr. Douglas Oard and Dr. Kevyn Collins-Thompson who were my mentors during the SIGIR 2016 Doctoral Consortium [23] for their valuable feedback provided. Each member of the Document and Pattern Recognition Lab (DPRL) at Rochester Institute of Technology (RIT) that very often provided useful feedback and new interesting ideas that helped me improved my work. Danilo Dominguez, Marien Ochoa and other PhD students friends who also provided valuable feedback. This material is based upon work supported by the National Science Foundation (USA) under Grant No. HCC- 1218801.

Dedicated to my parents Gladys and Jaime, my siblings Jessica and Alexander in Honduras. I also thank my aunt Nelly and uncle John in Allentown, PA for their continuous support through these years. I also dedicate this work to my dear friends that I met in Rochester, and of course to my lifetime friends back in Honduras that gave me their support along the way.

Contents

List of Figures

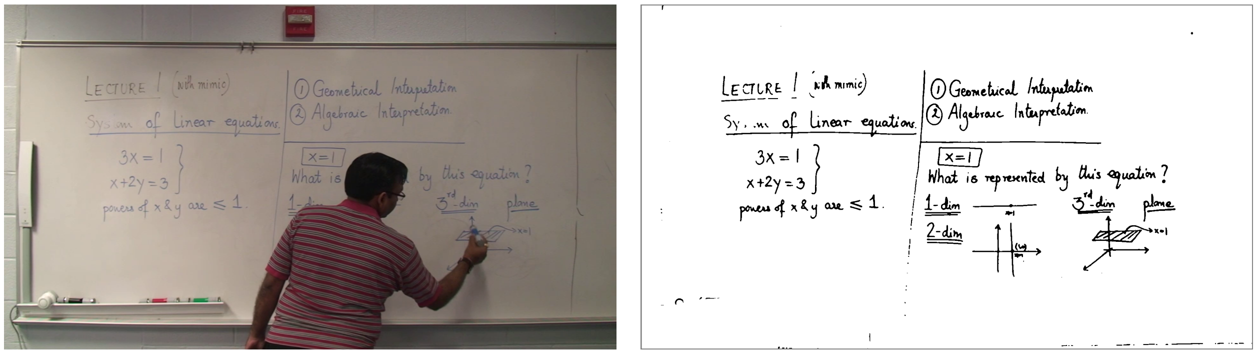
List of Tables

Chapter 1

Introduction

Mathematical expressions can be found within many collections of data including papers, encyclopedias and lecture videos (see Figure 1.1). A student might want to find papers describing algorithms with worst-case complexity $O(*log(*))$ where $*$ can be replaced by any variable. Another student could be watching a math lecture video where the speaker wrote "Recall $A\bar{x} = \bar{b}$" and wants to find the excerpt in a video collection where this formula appears for the first time. Most popular search engines today provide limited support for such information needs. Based on the notion that visually similar mathematical expressions are likely to be related, we developed scalable models for appearance-based retrieval of mathematical notation to fulfill information needs like the ones previously described. An alternative model is also proposed for cases where the semantics of a math expression are given by operator trees, and we show that improved formula retrieval results are obtained when they are combined with the visual representations.

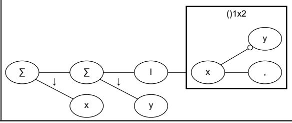
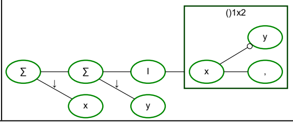
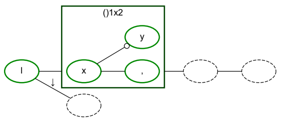
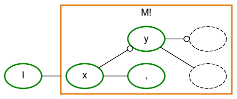
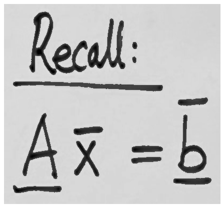
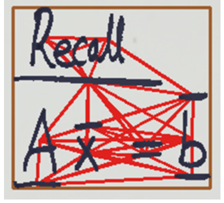
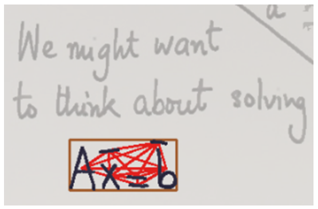
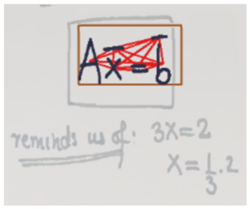
A single math formula retrieval framework is proposed for two different modalities: Symbolic and Image-based. The first one refers to the case where symbols are known and their structure (visual or semantic) is given. An example of Symbolic formula retrieval is providing the query formula $\sum_x \sum_y I\,(x,y)$, related to "Image Moments", as a LaTeX string and retrieving documents containing similar formulae in various formats like LaTeX or Presentation MathML (see first row of Table 1.1). Queries and formulae in the collection are converted to one of our internal repre-



(a) Raw video frame (b) Indexed key-frame

Figure 1.1: Example of input data for image-based formula retrieval. (a) Raw frame from lecture video. (b) Same key frame after binarization and content extraction.

Table 1.1: Symbolic vs Image-based formula retrieval. In the first row, a LaTeX query is provided to a Symbolic formula retrieval system [138], which finds an exact match and another two formulas containing sub-expressions of the query. The graph nodes marked green represent matched symbols while dashed nodes are unmatched symbols. In the second row, the query is a whiteboard image which refers to "system of equations" in the source video. Using an image-based formula retrieval system, we find some matches including the source as well as segments from other videos related to systems of equations matching part of the query.

	Query	Candidate Matches		
Symbolic	$\sum_x \sum_y I(x,y)$ 	$\sum_x \sum_y I(x,y)$ 	$I_p(x,y) = 1$ 	$I(x,y,t)$ 
Image-Based				

sentations, either Symbol Layout Tree (SLT) [138] or Operator Trees (OPT) [26]. Candidates are selected if their structure matches parts of the query graph. In Chapter 3 we describe our approach for symbolic formula retrieval which achieves state-of-the-art results with fast retrieval times and scalable index sizes.

The second modality, Image-based formula retrieval, is harder because only images of math expressions are given while the symbols and their semantic relationships are not provided. An example of this modality is providing a query image including a formula related to "systems of equations" and retrieving math lecture videos containing the same formula in segments where systems of equations are discussed (see second row of Table 1.1). In this case, we also assume that our index stores images containing non-delimited math expressions mixed with text and figures like the one shown in Figure 1.1(b).

Extracting the math expressions found in lecture videos is a challenging problem. The original raw video images need to be binarized and the handwritten content on the whiteboard needs to be separated from other elements in the image. In Chapter 4 we propose a novel lecture video summarization approach that, given lecture videos recorded with a still camera in the classroom, is capable of automatically extracting the handwritten content on the whiteboard as a small set of binary key-frames where the speaker and the background elements have been mostly removed from the images. The extracted images are ready for indexing and retrieval using an image-based approach.

The symbols and visual structure in images of math expressions can be identified using optical character recognition (OCR) techniques, segmentation, and parsing. However, this task is very difficult when the math expressions are handwritten and state-of-the-art methods for full recognition of math formulas have low accuracy. For example, consider the Competition on Recognition of On-line Handwritten Mathematical Expressions (CROHME) [77] where the best system achieved only 67.65% correctly recognized isolated expressions if the symbol classes need to be classified, and up to 84.38% if the correct symbol labels are known beforehand. Such a recognition rate is rather low considering that input data are traces of isolated expressions drawn on a tablet, and temporal information of the tracing process is available. In our case, the content recognition problem is harder because the handwritten math expressions are not isolated as shown in Figure 1.1(b). For this reason, we only rely on isolated symbol recognition results for which our own math symbol recognition approach described in Chapter 5 achieves competitive recognition rates on the CROHME datasets.

Finally, in Chapter 6, we propose our method for image-based formula retrieval, which avoids full recognition of the structure of math expressions, and is able to achieve promising results for image-based formula retrieval purely based on the appearance of math expressions.

Appearance-based math formula retrieval models are expected to have some limitations. For example, a purely visual approach could miss that two expressions are equivalent based on mathematical identities or notation usage. Also, math expressions are polysemic which means that identical formulas might represent different concepts depending on the context in which they appear [137]. In handwritten math expressions, the appearance of math symbols varies between writers. Despite these limitations, these retrieval models can be expanded, allowing users to find non-identical matches through partial similarity matching, variable substitutions and wildcards. Consider the query $x^2 + y^2 = *$, the candidate $\alpha^2 + \beta^2 = \gamma^2$ becomes a non-identical perfect match if the wildcard matches γ^2 and the variables x and y are replaced with α and β respectively. This type of non-identical match can be relevant for users browsing through formulae in Wikipedia, for example.

Our models are adaptable to other domains such as chemical or technical diagram retrieval where the assumption that visually similar elements are often related holds. For example, two diagrams of circuits might contain identical portions reflecting common modules that they share, and for some applications it might be useful to retrieve such diagrams using an image of the common module. Overall, the main requirement for using our model on a new domain is a graph-based representation that can be built reliably from the input data.

In real life applications, a math-aware search engine handles queries that might contain mixtures of keywords along with math expressions. Matching formulae in documents is a complementary tool to traditional text search engines. To know what is the best way to handle the interaction between text and math expressions remains an open question.

The main **research questions** that lead us to the proposed methodology in this work are:

1. What are the visual patterns that make us perceive two formulae as being similar?

2. How can we quantify these patterns using similarity metrics?

3. What are the most effective graph-based representations for formula retrieval in terms of primitives (nodes) and graph family (edges)?

4. How do we ensure consistency in graph representations for Image-based formula retrieval when binarization errors are present on the image?

Thesis statement: High quality formula retrieval results can be obtained by identifying candidates using symbol pairs in a query formula, and then re-ranking candidates by their structural similarity to the query.

Contributions: In terms of symbolic formula retrieval, we proposed the novel *Maximum Subtree Similarity (MSS)* metric for ranking formulae using the best query match with unification and wildcard support, and we achieved state-of-the-art performance on the NTCIR-11 and NTCIR-12 formula retrieval benchmarks, indicating that our approach is efficient in terms of time and space, and produces effective formula search results. We also proposed a way to combine both formula representations (Visual SLT and Semantic OPT) to obtain improved search results, and we provide some new insights for various similarity measurements between mathematical formulae for ad-hoc query-by-expression.

We also propose a novel approach for retrieval of mathematical notation from images that works with probabilistic math symbol recognition and does not require parsing the structure of math expressions. We provided preliminary results that show that cross-modal search between symbolic expressions and handwritten index image (and vice-versa) is possible using the proposed method.

We also contribute a novel method for lecture video summarization based on analysis of conflicts between stable connected components that appear on binarized images from sampled video frames, which allows us to extract the handwritten math expressions that appear on lecture videos. The method generates a spatio-temporal structure that contains very detailed information about the video allowing for very dynamic summaries. Using these key-frames and the original spatio-temporal structure, we have created our own navigation tools that let the user click on particular connected components of the key-frames to jump to the video segment where that particular element was written on the whiteboard.

We also provide a new dataset with 12 fully labeled lecture videos that can be used for benchmarking of approaches for binarization and summarization of lecture videos in the future. Finally, we proposed a shape-based feature set for recognition of on-line and off-line handwritten math symbols along with a method for synthetic data generation for math symbol recognition.

The source code of our symbolic formula retrieval approach is publicly available[1]. Our initial math symbol recognition code is also publicly available[2], and the data labeling tools, labeled videos and source code for image-based formula retrieval approach will be released in the near future as well.

In the next Chapter (Chapter 2) we present related work that informed our approach. This includes previous work in Mathematical Information Retrieval (MIR), Content-Based Image Retrieval (CBIR), mathematical symbol recognition, and video lecture summarization.

[1]`https://cs.rit.edu/~dprl/Software.html\#tangent`

[2]`https://cs.rit.edu/~dprl/Software.html\#crohme`

Chapter 2

Background

Multiple techniques are required to successfully achieve indexing and retrieval of mathematical notation. Many of them will vary according to the search domain. For example, the preprocessing required to index the math notation found in a web document in LaTeX or MathML format will be very different from the preprocessing required for lecture videos. Also, different indexing techniques will be required depending on the amount of information available in each domain. In this chapter we consider four major fields that are related to preprocessing, indexing, and retrieval of mathematical formulae from both web documents and math lecture videos, and these fields are: Mathematical Information Retrieval (Section 2.1), Content-Based Image Retrieval (Section 2.2), Lecture Video Summarization (Section 2.3), and Mathematical Symbol Recognition (Section 2.4).

First, Mathematical Information Retrieval is the main field of this research and covers the main principles of indexing and retrieval of mathematical expressions found in the literature. Second, Content-Based Image Retrieval is relevant to our research work since a part of our work, retrieving math formulae from videos, is an application of using images as queries to retrieve related images for a very restricted domain. Third, Lecture Video Summarization is relevant since we need to apply a method to extract and summarize the handwritten content from the whiteboard in lecture videos in order to index and retrieve such content. Finally, math symbol recognition can be useful for preprocessing of collections in cases where the math expressions need to be recognized for indexing, and it is also an important component of interactive search interfaces like m_{in} [102] that let the users draw the mathematical expression they want to look for.

2.1 Mathematical Information Retrieval

The field of Mathematical Information Retrieval (MIR) is relatively new [137]. Applications in this field aim to retrieve different types of elements that contain mathematical notation, and in many cases they use math expressions as part of the queries. Practical systems for MIR are few in number [58], but the number of situations where MIR can become a very useful tool is large and keeps growing. For example, we can use a traditional text-based search engine to find articles related to the Pythagoras's theorem using the keywords "Pythagoras" and "Theorem", but if we use a math-aware search engine then we could query the system using the formula $a^2 + b^2 = c^2$ to find additional articles that use Pythagoras's theorem without mentioning it explicitly. Additionally, as some authors have pointed out before [133], math-aware search engines can be great tools for mathematicians by helping them find existing proofs as they work on their own proofs. Math search engines can be used to find connections between manuscripts containing math, and even to find

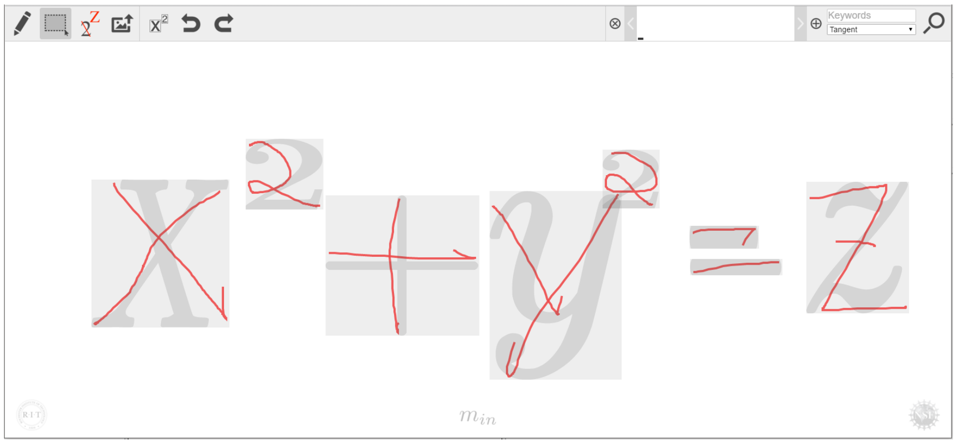


Figure 2.1: Example of the m_{in} interface [102] in edit mode. The user draws an expression and the system recognizes the math symbols, parses them, and converts the whole expression to LaTeX.

relationships between different fields of study [133].

Traditional text-based search is inadequate for retrieval of Mathematical Information [137]. According to Miller and Youssef [71], text-based systems have some important limitations. Math expressions include combinations of non alphabetical symbols which many text-based systems ignore or cannot handle correctly. Also, the structure found in math expressions delivers meaning and it is hard to encode and match reliably using strings. Another important element in math is the existence of equivalence relationships such as trigonometrical identities. This is similar to the concept of synonyms in text, but it is not feasible to simply store all possible equivalent expressions using a thesaurus the way it is done for a typical text-based application because a single expression has an infinite number of trivially equivalent expressions.

Like any other information retrieval domain, the query language is a very important element of any MIR system. MIR systems require flexible query languages that are natural and easy to specify for its users. Users might be required to write their queries using specialized notation like LaTeX or MathML, but practical systems should provide good user interfaces that facilitate this process. The usage of tools for handwritten math expression recognition can greatly help the creation of such interfaces. An example is the Mathematical Interface m_{in} [102, 139] (see Figure 2.1) that allows users to draw math expressions which are then recognized and converted to LaTeX and can be used as queries for multiple search systems including Tangent [112].

Independently of the input format, a good query language should include wildcards to make it easier for the user to find math notation without the need of being specific all the time. The standard definition of wildcards used for traditional text-based search systems is not adequate for MIR [4]. In the work by Altamimi and Youssef [4], three different types of wildcards for MIR are described which we cover later in Section 2.1.1.

In his work [133], Youssef describes some of the basic and long terms goals of math search. The first goal is math awareness which refers to the system capabilities for recognition of mathematical symbols and structures. Second is natural math-query language, which refers to the ability of accepting queries in a format that is common to its users, like for example accepting simple LaTeX strings that many users know how to write. Third is fine granularity of searchable and retrievable information units which refers to the minimal retrieval unit used by the system where smaller is generally better. Fourth is perfect recall, and it means retrieving all relevant matches for a given query. Fifth is perfect precision, and it means retrieving only relevant matches. Sixth is perfect relevance ranking which means to prioritize query matches by relevance to the users. Seventh is useful highlighting which is allowing the user to know what parts of a query are matched by a given result. Finally, the eighth is minimum hit-redundancy which means to avoid presenting to the users multiple instances of the same match as independent matches because they are redundant. Most MIR systems should attempt to achieve these goals to some extent, and in our work we have achieved many of them already (math-awareness, natural math-query language, fine granularity of searchable and retrievable information units, and useful highlighting), while we have taken steps toward achieving the rest of them (perfect recall, perfect precision, perfect relevance-ranking, and minimum hit-redundancy). In the long term, math search can be useful to discover similarities between fields, computer assisted prof creation and learning aid.

Interest in Mathematical Information Retrieval (MIR) has increased in recent years, as witnessed by math retrieval tasks at the NTCIR conferences [2, 3, 136]. These competitions provide a common ground to evaluate and compare different approaches using medium and large datasets and fixed sets of queries. The MathIR arXiv Main Task at NTCIR-12 [136] consists of finding the most relevant documents from the arXiv[1] collection for a set of 29 queries containing keywords and math expressions. The optional MathIR Wikipedia Task includes 30 queries containing keywords and math expressions, and the optional MathIR Wikipedia Formula Browsing Task includes 40 queries for isolated math expressions. Participating teams submitted up to four runs for each task, and the top-20 results from submissions for each query received relevance assessments from evaluators. The data that was produced for these tasks will be very useful in further advancing the field of Mathematical Information Retrieval, and we use this dataset in our experiments in Chapter 3.

In the following sections, we will focus on the Formula Retrieval components of existing MIR systems. We distinguish two major modalities of the formula retrieval problem: Symbolic and Image Based. The first one refers to the case where symbols and structure are known as in the case of LaTeX expressions or MathML. The second one refers to the case where only images of the math expressions are provided and the symbols and their relationships are initially unknown.

2.1.1 Symbolic Formula Retrieval

Symbolic formula retrieval methods work with documents containing math notation represented as text. These representations encode the hierarchical nature of math expressions as strings. Queries can also be provided as strings, but it is possible to use math recognition systems that build a compatible representation from traces or images of math notation for search like m_{in} [102]. During the preprocessing phase, multiple approaches will detect existing expressions on each document in the input collection, and then each expression will be converted to the internal representation of the system and will be stored in the index. Some approaches add normalization and canonicalization steps where expressions are transformed to guarantee that certain non-identical equivalent elements will be matched as equal. For example, the expressions $x + y + 1$ and $y + 1 + x$ are equivalent and both could be matched by the query $1 + y + x$ if a canonical representation is used.

There are multiple ways to perform formula retrieval when symbols and their relationships are known. In this subsection, we will concentrate on three major aspects of methods for symbolic formula retrieval: representation, indexing and ranking.

[1]http://arxiv.org/

$(x+y)^2$

(a) Expression

`$ { \left( x + y \right) } ^ 2 $`

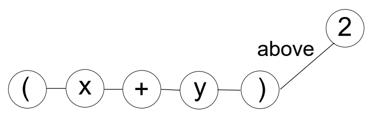
(b) LaTeX

pow add x y 2

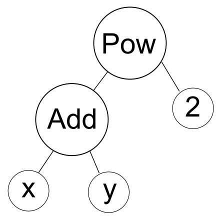
(c) Flattened

```
<math>
     <msup>
         <mrow>
              <mo>(</mo>
              <mi>x</mi>
              <mo>+</mo>
              <mo>y</mo>
              <mo>)</mo>
         </mrow>
         <mn>2</mn>
     </msup>
</math>
```

(d) Presentation MathML



(e) Symbol Layout Tree



(f) Operator Tree

pow add x y 2

add x y

x

y

2

(g) Tokens

pow add id1 id2 const1

add id1 id2

id1

id2

const1

(h) Generalized tokens

Figure 2.2: Examples of different representations used for math expressions. (a) is the original expression. (b)-(d) and (g)-(h) are string representations. (c) is a type of flattened string representation using prefix notation. (e) and (f) are tree-based representations. The provided tokens are just examples, and different string tokens have been proposed in literature.

Representation

Symbolic formula retrieval systems usually handle collections containing formulas in a text-based representation like Presentation MathML, Content MathML or LaTeX. Content MathML is considered a semantic representation of math expressions because it provides enough metadata to build operator trees [37]. On the other hand, presentation MathML and LaTeX are considered appearance-based because they only describe the symbol layout of math expressions. Figure 2.2 contains examples of different representations for math expressions.

One common issue of appearance-based representations of math expressions is ambiguity. Given a math expression, it is ideal to interpret it according to the meaning intended by its writer. However, math notation can be ambiguous and sometimes the right interpretation of a math expression becomes context dependent. For example, $f(x+y)$ is usually interpreted as the function f with the result of $x+y$ as input value, but it is also possible to interpret it as the multiplication of the variable f by the sum of x and y.

Some studies attempt to define which of the two basic representations, semantic-based or appearance-based, is better for MIR [80]. While semantically enriched representation of math expressions are theoretically better for MIR, most published material encodes math expressions only in terms of their appearance [47], and methods that only work for semantic representations will not be able to index such material properly. It is possible to convert data from one representation to the other, but his is typically an error prone process because in many cases, it is hard or impossible to infer the semantics of a given notation without contextual information that defines the meaning of the symbols being used. Many approaches participating in NTCIR-12 MathIR tasks [136] used a combination of both representations.

Independently of the input representation, most MIR systems add preprocessing steps that will transform the original math expressions to a different representation for their index. Most representations fall within one of the following categories: text-based, tree-based, and spectral.

Text-based representations. These use strings extracted from the original format to represent math expressions. Flattening is required to encode the structural component of math expressions into strings (see Figure 2.2(b)). Approaches using this type of representation tend to be limited in practice, and work well mainly to find exact matches of a query. To increase the likelihood of finding matches, some methods use canonicalization to simplify expressions, and to identify commutative operators and equivalences [59, 99]. It is also common to enumerate identifiers to support generalized variable matching and/or unification [33, 99, 116, 135]. Some existing approaches include the seminal work by Miller et al. [71] for search of expressions in the Digital Library of Mathematical Functions (DLMF) using LATEXstrings, and the work by Mišutka et al. [72] which flattens expressions using postfix notation for math search. One advantage of converting math to text is that existing optimizations in text search engines such as ranking by TF-IDF [71], topic modeling, and word embeddings [116] can be used.

Tree-based representations. These approaches consider the structure of math expressions and index formulas as complete SLTs or OPTs, and tend to be more adequate than text-based representations search. Typically, the hierarchical structures in formulae are mapped directly, and organized within tree-based indexing structures [37, 47, 145]. In some of these approaches, all subexpressions in formulae are indexed to support partial matching, with common subexpressions labeled and shared to reduce index sizes [47]. In some systems, semantic or operator trees (see Figure 2.2(f)) are built from Content MathML [37, 49, 50]. However, appearance representations are common in large collections and multiple systems work with various layout tree representations extracted from LATEX strings [104] and Presentation MathML [47, 52, 60, 110, 138] (see Figure 2.2(c)-(d)). Some authors have made comparisons of the performance of their systems using both semantic and appearance-based representations [80, 100], and conclude that semantic representations should be preferred when they are available. Finally, hybrid approaches exists that use presentation MathML enriched with semantics [32, 58, 125]. In our case, we shown through our experiments in Chapter 3 that better results are obtained if both semantic and appearance-based representations are available and used together.

Spectral or feature-based representations. These methods extract some high level features from the original format and use them to represent math expressions. The main insight of these approaches is that simpler primitives allow more partial matches, increasing recall. Path-based methods store sets of paths from the root to internal nodes [38] and leaves [145]. Paths may reflect operator commutativity by inserting symbols [145] or using unordered paths [51]. Some use hashing to encode subtrees [51, 85]. In the work by Nguyen et al. [82], finite state machines are used for extraction of text features from Content MathML. A different approach also by Ngugeyn et al. [83] uses text features to build lattices of math expressions. Pinto et al. [93] use strings of features for math expression representation. Our Tangent formula retrieval models described in Chapters 3 and 6 fit within this category as they use a spectral representation of math expressions based on tree symbol pairs along with their relative paths for indexing and initial retrieval.

Indexing

The math expression indexing method of a MIR system is fundamental because it defines the initial candidate matching procedure and determines query execution time. Many systems take advantage of existing optimizations for text-based search engines and they simply encode math expressions using strings that can be stored and retrieved using a regular text engine. Common text search engines used by MIR systems include Apache-Lucene [2], Solr [3] and ElasticSearch [4]. Other systems use their own custom engines optimized for tree or spectral representations. One advantage of using a text search engine to store math expressions is to keep text along with formulas in the same index.

There are multiple ways to encode math expressions using strings for indexing with a text engine. One option is to flatten the expression to convert it from the original representation into one or more strings that represent the same expression in a format that the text engine can handle [71, 72] (see Figure 2.2(b)).

A common approach for systems using tree-based representations of math expressions is to apply a tokenization process to represent each tree using multiple text tokens (see Figure 2.2(g)-(h)). The token extraction process is usually different between systems and affects the performance and matching capabilities of each system. Sojka et al. [100, 110] use a compact representation of Presentation MathML to create tokens for each expressions at different depth levels in the tree. MCAT and other systems [52, 80] extract tokens from Presentation MathML based on three types of paths: Ordered paths (opaths), Unordered paths (upaths) and siblings (sisters). The WikiMirs systems [32, 58, 125] builds semantic trees from enriched presentation MathML, and then indexes original and generalized tokens (see Figure 2.2(g)-(h)) representing full subexpressions at different depths of the semantic trees. Lipani et al. [60] also uses generalized tokens to represent math expressions on their index. Finally, in the approach by Pinto et al. [93], strings of features are extracted and stored using the ElasticSearch engine.

Methods that use the substitution tree representation also build their index using the same trees. Examples of this approach are the Math Web Search systems [37, 49, 50] that built substitution trees from Content MathML. Another example is the work by Schellenberg and Zanibbi [104] who use Substitution Trees based on layout instead of semantics.

Systems that use a spectral or feature-based representation of math expressions usually build their own custom indexes. In the work by Nguyen et al. [82], formulas are indexed based on a set of features. In a more recent approach, Nguyen et al. [83] use features to create a lattice-based index of math formulas. Kamali and Tompa [47] treat the node labels of a tree-based representation as bags of words and build an inverted index for fast candidate selection.

Both the original Tangent formula retrieval model [112] by Stalnaker and Zanibbi, and its second version Tangent-2 by Pattaniyil and Zanibbi, use symbol pairs extracted from a Symbol Layout Tree representation. Two elements are added to the pair to describe the horizontal and vertical distance between the two symbols [88, 112]. Our proposed model, Tangent-S, is an extended and improved version of these models where relative paths between symbols in the SLTs are used instead of the horizontal and vertical distances.

[2]http://lucene.apache.org/
[3]http://lucene.apache.org/solr/
[4]http://www.elasticsearch.org/

Ranking

A good retrieval system produces rankings that satisfy most user queries by listing the most relevant documents first. When exact matches of a query are found in multiple documents, retrieval systems need to prioritize those that are believed to be more relevant to the current query. Math notation is ambiguous by nature, and context is required to understand some expressions [137]. Sometimes an exact match for the query is found in a context that makes it irrelevant to the user. For example, simple patterns containing common variables, like $x + 1$, might appear in multiple documents and the same variables might refer to different concepts on each document.

In multiple applications of MIR, search for approximate expressions is required. Using the right canonicalization and normalization techniques multiple approaches can successfully locate exact matches for queries. However, certain variations might be introduced that are harder to handle through preprocessing. Sometimes the users might not remember the exact formula they need to find. In these cases, approximate search becomes a great tool to help users find a wider range of similar expressions, and sometimes exact subexpressions of the query. Approximate search requires the usage of similarity metrics that predict what users might perceive as similar to a given query. Such similarity metrics also play an important role in ranking documents.

In this work, we are more interested in similarity metrics for isolated math expressions. However, many of the MIR approaches discussed here do not distinguish between math formula and text search, and the final metrics proposed by them can only be used to rank entire documents. Such similarity metrics attempt to approximate the relevance of math formulae in the context in which they appear and do not consider partial similarity between isolated formulas.

Notions of similarity between math expressions. According to Youssef [133], tree representations can be considered to be the basis for defining similarity functions. In general, the more that two formula subtrees have in common, the more similar they become. Assuming a semantic tree representation, the author also suggests that matching internal nodes of the trees is more important than matching the leaves, which tend to be variables names or constants that have less impact on the operations and structure of the math expression. For example, in the operator tree for $(x + y)^2$ shown in Figure 2.2(f), we could swap the labels of the leaves and the resulting expression $(y + x)^2$ is equivalent to the original, but if we swap the labels of the two internal nodes we obtain a radically different math expression $(x^y) + 2$.

In different work, Youssef [134] discusses concepts about relevance ranking for math search. Metrics like term frequency - inverse document frequency (TF-IDF) that are widely adopted for text search tend to produce bad results for math search. The relevance of a math expression changes with its context, and there are dynamic and static components of the relevance of a math expression. Dynamic elements are query dependent while static components are query independent and can be determined at index time. Some static elements of relevance include the types of the math object (operator, variable, function, etc), the categorical importance of terms (in many cases operators are more important than variable names), and the number and types of cross-references between documents and math expressions. Most of these ideas are in agreement with what is proposed later in this document. For example, the author proposes the usage of relevance vectors for math expressions and the usage of two-stage retrieval for efficiency which are common in other domains in the information retrieval literature. Our Tangent-3 model described in Chapter 3 adopts the usage of multiple scores for math expression re-ranking on a three-layer formula retrieval model.

Schubotz et al. [106] evaluated a set of similarity factors for math expressions. The five main concepts explored that might affect similarity between math expressions include: taxonomy of functions, hierarchies of data types, match depth, query coverage and formula versus expressions.

The taxonomy of functions can be used to define the class of a given function and it is assumed that two functions are more similar if they belong to the same class or mathematical domain. For example, a function containing matrices is likely to be considered similar to other function from the same class of linear algebra functions containing matrices. The data-type hierarchical levels assume that some math objects are more important than others at the time of measuring similarity. According to the authors, we can define four levels: 0 - contants, 1 - variables, 2 - functions and operations, 3 - functionals like integration and differentiation. It is suggested that math objects of a higher level should have larger weights when evaluating similarity. The depth of a match refers to the location of the common elements between two expressions. In this case, the authors suggest that deeply nested matches have lower relevance for a query. Query coverage assumes that the more terms and structure that two terms share, the more similar they are. Finally, formula versus expression assumes that formulas (math expressions containing a comparison operator) are more relevant than non-formula expressions. However, in the final evaluation step, the authors re-used data from the NTCIR-11 competition which contained relevance assessments for entire documents containing multiple formulae using queries that also contained keywords. Ideally, these concepts should be evaluated using relevance data for isolated math expressions, because this will ensure that any observed differences in relevance between candidates for a given query are more likely to be caused by the factors being studied and are not the product of some other factors that are not being studied directly.

Approximate Matching: Unification and Wildcards. Multiple methods allow users to find fuzzy matches for their original queries which means finding non-identical math expressions that could be relevant for a given query. In other cases, systems allow users to search over broader ranges of math expressions by using wildcards. In multiple contexts, a wildcard can save user time by not having to type a full formula or make the search process easier when the user does not know the exact math expressions that he/she is looking for. Some systems provide functionalities for unification of symbols and wildcard matching, but sometimes these matches are accepted as valid with penalties based on how different they are from the original query [32, 47, 138].

Unification of variables and/or constants can be implemented in multiple ways and some approaches index math expressions in a way that facilitates finding unifiable matches. In particular, approaches that use text search engines tend to use modified or generalized tokens that can be matched by multiple unifiable expressions [32, 58, 72, 110, 125]. Canonicalization of the math expressions is also applied by some of these approaches to ensure that generalized tokens can be matched more easily.

In the case of approaches that implement tree-based matching of math expressions, the unification process can be more detailed and flexible. Approaches using Substitution trees [37, 49, 50] work directly with hierarchical structures where expressions with the same structure but different operands will share common ancestors. The concept of unification, allowing substitution of operands and/or operators, is very natural for these approaches. Other methods use tree-based similarity metrics that consider unification during retrieval time. In the work by Kamali and Tompa [47], the tree edit distance is used for similarity matching, and unification of variables is treated as an edit operation where labels of the tree get replaced with a score penalty.

Wildcard matching is another important component of retrieval systems. Similar to wildcards for text search that can be used to match arbitrary substrings at given locations, wildcards in math search let the user find expressions that can contain arbitrary subexpressions at given locations. One important difference is the tree structure of math expressions that needs to be taken into account by wildcard matching implementations. Note that implementing the full functionality of wildcards can be impractical. A single query may contain multiple wildcards, and each could be expanded to match any arbitrary subpattern. If the number of math expressions in the index is very large, then finding all possible matches for a wildcard pattern might take too long to be useful in real applications.

In the work by Altamimi et al. [4], the authors define three types of wildcards required for math retrieval systems: term-level, component-level, and bounded. The term-level wildcard can be used to match terms or subexpressions at given locations of the query. For example, they can be used to match a single argument in a function with multiple arguments, but that argument can be a long sub-expression. The component-level wildcard is more abstract and can be used to match multiple terms. For example, they can be used to match all arguments in a function with multiple arguments, or even all elements within a matrix. The last type is the bounded wildcard and can be of any of the two previous types, but it has a name and has the additional restriction of being bounded to other wildcards sharing the same name. This type of wildcard can be used to match arbitrary patterns at multiple locations in a expression, but forces them to be identical or different as specified in the query. For example, if we use the notation $*x*$ to define wildcards with name where x is the name of the wildcard, then we can use the query $f(*1*) = *1* + *2*$ to specify that we want all functions f of any variable name $*1*$ that are equal to the sum of the same variable $*1*$ plus another subexpression $*2*$ which at the same time has to be different from $*1*$, this means that $f(x) = x + 1$ and $f(y) = y + \frac{\pi}{2}$ will match the query but $f(a) = a + a$ should not.

Some MIR systems that consider wildcards only support single symbol replacement while other systems provide greater functionality by allowing matching of sub-expressions. Some approaches use generalized tokens on the index to allow wildcard matching with single symbol replacement [60, 71]. The Tangent 2 system [88] implements wildcard matching by expanding generic tuples in the query that contain wildcards by finding concrete tuples that match the given pattern. In the end, the system is only able to do single symbol replacement.

Other systems implement wildcards allowing subtree matching. Methods based on substitution trees [37, 49, 50] use an index method that already considers that multiple math expressions share a common generic pattern by replacing nodes with arbitrary sub-expressions in the hierarchy of substitutions. Finally, in the work by Altamimi et al. [4], wildcard match is implemented for XML representations of formulae like Presentation or Content MathML using XPath/XQuery engine. However, the XQuery language does not provide ways to define wildcards bounded by name.

In Tangent-3 described in Chapter 3, we start with single symbol matching for wildcards at the first layer of our retrieval model, followed by a very detailed wildcard matching algorithm in the second layer (see Section 3.3.2) that allows matching term-level, component-level and bounded wildcard types as described in the work by Altamimi et al.[4].

Ranking methods. Every retrieval system needs a method to rank the matches found for a given query. In this section we only discuss methods that consider approximate matches and that require similarity functions to rank these non-identical matches.

Many methods that employ text search engines for indexing and retrieval of math expressions apply traditional text retrieval metrics for similarity ranking [52, 60, 80, 93, 125]. Other methods also based on text engines propose novel token weighting schemes designed specifically for math expressions [32, 52, 58, 72, 100, 110]. EgoMath2 uses generalized tokens in the text-based index to prioritize operators over operands in the similarity ranking [72]. The MIaS system assigns weights to each token based on the depth of the tree from which it was generated [100, 110], and the method has been tested using both Presentation MathML and Content MathML [100]. The MCAT system [52] uses traditional TF-IDF in the first retrieval step and then reranks math expressions using a linear combination of content similarity and structure similarity. The MCAT system gives stronger weights to operators and numeric literals and weaker weights to identifiers. The Wikimirs 2.0 system [58] considers independent scores for similarity between the query and formula, and composite scores between the query and all relevant formula in a document. Their scoring method still uses statistics of term frequency. The ICST retrieval system [32] uses term frequency - inverse formula frequency (TF-IFF) considering penalties for matches against generalized tokens.

Other recent approaches are no longer considering formulas as separate entities, but they also consider the context in which a formula is found. They index additional text tokens extracted from the context of a math expression [125]. These extra tokens can be used for search including keywords, but they can also be used to determine relationships between math expressions.

In the work by Kamali and Tompa [47], tree edit distance is applied to evaluate the similarity between math expressions using a layout-based tree representation of formulae. Since this metric is expensive, the system considers early termination techniques to avoid computing tree edit distance for expressions that are unlikely to be relevant, and it also builds a cache with memoization to avoid computing the tree edit distance between common subtrees multiple times.

In the work by Nguyen et al. [82], two types of features are extracted from Content MathML representations: single features (constant numbers, variable names, function names, etc) and combination features (combinations of math operators and operands). These features are then used to train a Passive-Agressive algorithm for learning to rank with ranking perceptron optimization to rank documents. In a different work, Nguyen et al. [83] propose the usage of a concept distance metric to measure similarity using a Lattice of math expressions.

The Tangent [112] and Tangent 2 [88] systems use the Dice coefficient of pairs of symbols matched between symbol layout trees. A tuple was formed by using every two symbols that have an ancestor-descendant relationship in the symbol layout tree, and each tuple adds to the two symbols the horizontal and vertical distance between them. Additional methods and similarity metrics for math recognition and retrieval are covered in the survey by Zanibbi and Blostein [137].

2.1.2 Image-Based Formula Retrieval

Compared to symbolic formula retrieval, there are very few methods proposed for math formulas retrieval directly from images. Existing approaches have very little in common, and there are no standard datasets that can be used to compare them. In particular, we are interested in robust methods that can be applied for retrieval of noisy images of handwritten math expressions.

In the work by Marinai et al. [69], images of mathematical symbols are indexed using a bag of visual words approach. Keypoints of math symbols are described using shape contexts, and the visual dictionary is built using clustering through self-organizing maps. Similarity is measured using the cosine of the angle between weight vectors. In a more recent version [70], the earth mover's distance (EMD) is used instead of the cosine similarity. This method was designed and tested for retrieval of isolated math symbols, not complete expressions.

The method proposed by Zanibbi and Yuan [141] uses contour and density features extracted from the connected components (CCs) of math expressions images to match similarity. In a different work, Zanibbi and Yu [140] used dynamic time warping to align pixels projections of upper/lower image halves to compare similarity between images of handwritten queries and typeset document images after segmentation via X-Y cutting.

Chatbri et al. [18] present an approach for spotting queries based on a voting system where every connected component matched between the query and the candidate votes toward a location of the query in the image. Strong candidate locations with multiple votes are later identified as query matches. In a more recent approach, Chatbri et al. [17] use shape features of key-points extracted from foreground and background pixels to match binary images of math expressions.

All these methods described so far avoid recognition because they work with noisy handwritten input data. Optical character recognition (OCR) and other segmentation and parsing techniques might be used to convert these images into symbolic representations of math expressions. However, this task is very difficult when the math expressions are handwritten. For example, state-of-the-art methods participating in the Competition on Recognition of On-line Handwritten Mathematical Expressions (CROHME) [76] achieved only 62.15% correctly recognized isolated expressions. This is rather low considering that input data was online which means that tracing information is available instead of images which tend to be even harder to recognize. However, some methods have used content recognition in the past. In the work by Kanahori et al. [48], document images are processed to identify math content regions and these are recognized using an OCR system for math. We prefer approaches capable of working with noisy handwritten text, for example, in whiteboard images.

2.2 Content-Based Image Retrieval

The field of content-based image retrieval (CBIR) studies methods for search of images based on the actual content in the image pixels. It is always possible to retrieve images if they have meaningful metadata associated with them. However, many applications require finding images that have not been explicitly labeled by users. CBIR methods typically aim to solve this kind of application.

In this work, we briefly study general CBIR techniques because image-based formula retrieval represents a specialized type of CBIR. As explained in Section 2.1.2, using full expression recognition methods is possible but current approaches have very low precision and are not practical for search. Our method proposed in Chapter 6 uses recognition of math symbols only, but avoids recognizing the complete structure of math expressions.

Earlier approaches for CBIR measured similarity between images based on low level features like color histograms and textures [31]. In the case of binary images of whiteboard content, texture and color provide little information compared to spatial layout and shape of the content.

In math notation, the 2D location of each symbol defines part of its meaning in a given math expression. For this reason, in this work we will focus on CBIR methods that consider spatial layout. Many recent approaches still follow the general framework defined by the Bags-of-Features method [109]. Many recent methods add new optimizations to the basic framework and many consider efficient implementations of spatial constraints on their image matching system.

In the following sections we discuss the general bags of visual words approach [109], and describe some improvements proposed to the original approach. We also discuss the connection of sketches with image retrieval since most approaches for CBIR are designed and tested over natural scene imagery, but in our application we have binary images of math expressions where color and texture have little relevance, but shape and layout of the basic elements in the image define its meaning.

2.2.1 Bags of Visual Words

Many recent approaches for CBIR still follow the Bags-of-Features framework [109] also known as Bags of Visual Words (BoVW). Here we discuss only methods that consider the spatial layout of image content.

In the BoVW framework, relevant images for a given query image are selected by matching common descriptors between images. This operation can be expensive and unfeasible if it has to be

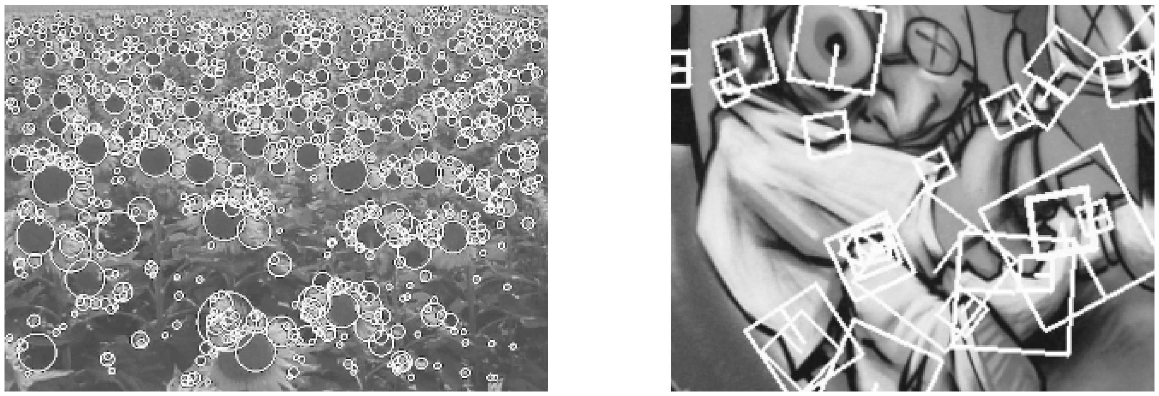

(a) Key-points (b) Patches described

Figure 2.3: Example of SURF local descriptors (Extracted from original paper [10]). (a) Key-points selected for a sunflower field (b) Size and and rotation of descriptor patches on graffiti scene.

performed over every single image in the collection. For this reason, BoVW methods create visual dictionaries that define clusters of local descriptors called visual words. Dictionaries define entries for inverted indices of images.

Given an image query, the set of visual words it contains are extracted and these are used to identify all images in the collection that share very similar local descriptors. The more visual words that two given images share, the more similar they are considered. After identifying an image as a candidate for a query, additional restrictions like spatial constraints might be applied to validate that it is a good enough candidate for retrieval.

Visual Words

Visual words represent the smallest matching unit in the BoVW approach. A patch of an image can be represented by a single visual word. These are based on local descriptors or features that describe the patch of the image from which they were extracted (see Figure 2.3). Local descriptors can be multidimensional or might require several bits to provide enough discriminative information about the patch they describe. However, thanks to the usage of visual dictionaries, they can be represented in a very compact manner using integers associated with their entry numbers in the visual dictionary.

The performance of the visual words approach is very sensitive to the discriminative power of the local descriptor used and the key-point selection method. Different local descriptors have been used to generate visual words including: SIFT [66], SURF [10], RootSIFT [7], USB [142] and others.

Note that depending on the nature of the data in the input collection, some local descriptors might be more adequate than others. For example, general descriptors like SIFT and SURF were designed for natural scenes imagery and work very well for that kind of images. However, when input images are binary, different local descriptors like USB [142] and Chatbri's [17] could be better choices. Features that were specifically designed for binary image tend to be more compact [142], and they choose key-points on the image in different ways [17] that could be more adequate for binary images.

Depending on how the local descriptor works, different key-point selections methods can be used to obtain the most discriminative but consistent set of local descriptors for a given image. Consistency of key-point selection methods is an important component that BoVW approaches require to work well. For example, given several images of one object viewed from different angles, BoVW approaches rely on the ability of their key-point selection method to consistently identify and extract visual words from the same salient points of the object in the images. Otherwise, there would be little or no overlap at all between sets of visual words for different images, and matching would become very difficult or even impossible.

Another important issue is that visual words are typically generated from quantized local descriptors. This, however, is prone to quantization errors and does not work very well for descriptors that fall close to the boundaries between clusters. For this reason multiple improvements have been proposed to mitigate these errors including techniques like soft-assignment [45, 92] and Hamming Embedding [45]. In the case of soft-assignment, a local descriptor can be assigned to multiple visual words using a fuzzy membership function [45, 92]. In the case of Hamming embedding [45], a hash-code is generated to refine the location of a quantized descriptor in the feature space, and hamming distance between hash codes will be proportional to the distance of the original descriptors in Euclidean space.

Visual Dictionaries

Visual dictionaries define the set of visual words that will be used for retrieval by a CBIR system. These dictionaries cluster entire regions of the local descriptor feature space into single unique visual words. Different techniques can be used to generate them depending on the underlying local descriptor selected. A simple approach is based on vector quantization, but data driven approaches based on clustering of local descriptors are also very common.

Multiple design decisions can affect the quality of the generated visual dictionaries and ultimately the performance of the CBIR system. The first option is the quantization or clustering method used to generate the visual words. Some traditional techniques like K-means and hierarchical clustering are widely adopted. In the work by Marinai et al. [69], self-organizing maps generate the clusters of local descriptors. One very important parameter is the size of the dictionary. A small dictionary will not be discriminative enough to differentiate between some images, but if the dictionary is too large then the resulting visual words might become too unique and similar shapes will not be matched. Depending on the local descriptor and the application, the sizes of visual dictionaries tend to vary from a few thousands words to more than a million of them.

In particular, approaches working with natural scenes tend to use very large vocabularies and do not consider that a single visual word is likely to appear multiple times on a single image.

However, certain images containing repeated patterns, like buildings with hundreds of identical windows, might have large numbers of instances of a single rare visual word which only appears in very few images of the collection. This is known as the “burstiness” problem [44]. Many BoVW approaches rely on TF-IDF metrics to give larger relevance to rarer visual words [44]. However, images with repeated patterns might contain many instances of visual words that are rare in the entire collection. In this cases, these rare visual words overweight other relevant patterns and cause some images to have overestimated relevance. Jegou et al. [44] propose dealing with this issue by considering local frequency reducing the relevance of visual words that appear multiple in the same image.

The target application of this work deals with binary images of whiteboard content. Visual words are expected to be repeated many times on the same image and the visual dictionary might need to be not too large compared to dictionaries for natural scene imagery.

Spatial Verification

Given a query image, candidates are selected by matching common descriptors. Some techniques have been proposed to ensure that two images not only share the same visual elements, but also that these share the same rough visual alignment.

In many approaches, candidates are located first without considering spatial arrangement, and the set of the most promising candidates are then reranked or filtered using spatial verification techniques. The RANSAC algorithm [91] finds correspondences between matching local descriptors very quick. It takes multiple random combinations of matches as reference points that define a transformation, and then evaluates how many of the remaining matches are consistent with the resulting transformation. RANSAC is useful for image alignment, but is also useful to verify the spatial consistency of matching points.

In general, points aligned from one image space to another image space define a transformation. In many cases, two images of the same objects can be aligned by transforming one image through rotation, scaling, and translation. These are affine transformations that can be represented through matrices in Hough space. Some approaches work by finding the most likely transformation from Hough space [8, 55]. In the work by Jegou et al. [45], weak geometric consistency is proposed which verifies how consistent are the transformations of matched local descriptors in terms of scaling and rotation. In the work by Zhang et al. [143]. a new type of topological verification is proposed that works for a wider range of elastic transformations by considering topology of matches instead of Hough space transformations.

While spatial verification during reranking is probably a good practice for efficient retrieval of consistent matches, it is also possible to encode additional information in the index to ensure that the set of initial candidates for reranking already have some spatial consistency with the query. For example, the visual phrases approach [144] stores small groups of visual words (phrases) in their index taking into account the local spatial arrangement between these visual words.

2.2.2 Sketches and Image Retrieval

Different shape-oriented methods have been proposed for sketch and handwriting retrieval. Sketches are basically handwritten which means they are bound to be noisy and imperfect. Sketches have no color or texture; they only have shape. There are different cases where sketches are considered in CBIR, and they include: sketch-based image retrieval, sketch-based sketch retrieval and handwriting image retrieval. In query-by-example systems, sketches can be provided as either traces (online) or images (offline). In this work, we are more interested in offline sketches since we will work with images of handwritten whiteboard content.

Sketch-based image retrieval uses handwritten sketches to search images in general. One example of these approaches is the work by Cao et al. [14], where sketches are used to query image collections. The authors define edgels as image edges at a given location with a given orientation. Edgels are extracted from each image in the collection, and these are stored in an inverted index of edgels. Given a query sketch, the system computes the edgels of the query and find similar images. One advantage of this method is that it is fast which makes it good for large collections. One major drawback is that edgels are defined at absolute positions, making it sensitive to affine transformations.

Sketch-based sketch retrieval uses handwritten sketches to search images of other sketches. In the works by Liang et al. [57] and Sousa et al. [111], handwritten sketches are described and matched at two levels, locally using some type of shape feature and globally using visual layout features. The method proposed by Liang et al. [57] has been tested for successful retrieval of technical drawings. The method by Sousa et al. [111] has been tested for retrieval of vector drawings [111].

Finally, handwriting image retrieval refers to methods that find images of handwritten text using feature-based matching instead of attempting to recognize the content first. Keyword spotting methods fall within this group of approaches. For example, in the work by Rath and Manmatha [97], discrete Fourier transform features and dynamic time warping are used for word shape matching along with K-means and agglomerative clustering for word spotting on historical documents.

2.3 Lecture Video Summarization

There are multiple reasons to create video summaries, like providing quick overviews that facilitate user navigation of videos. Summaries can also be used for indexing and retrieval of particular video segments based on their visual content. The survey by Hu et al. [41] provides additional details about many approaches for indexing and retrieval of videos. Detecting handwritten content in video images is a special case of text detection in imagery which is covered in the survey by Ye and Doermann [131]. Also, approaches for text detection, tracking and recognition in videos are covered in the survey by Yin et al. [132].

In order to retrieve mathematical notation from lecture videos, a video summarization approach is required to extract and summarize the handwritten whiteboard content for efficient search. Videos in this particular domain typically represent a single scene with one shot, but some might contain multiple shots when the focus shifts from the whiteboard to a person or an object in the classroom. In this section we cover some key ideas for video summarization, including key-frame extraction and evaluation methods, along with existing work in lecture video content extraction and summarization.

2.3.1 Key-frame Extraction

The problem of creating a video summary is closely related to key-frame extraction from videos. A very simple approach for video summarization is to compute key-frames of the input video and then select those that are the most representative of the entire video. A good set of key-frames has little redundancy and good content coverage [41]. Traditional key-frame extraction techniques are based on the analysis of different types of features like: color histograms, edges, shapes, optical flow, MPEG-7 motion descriptors, MPEG discrete cosine, camera activity and features derived from camera motion [41]. Among different approaches for key-frame selection, some use global comparison between frames to distribute the key-frames by minimizing an objective function that can be application dependent [41]. For example, the work by Li et al. [56] uses minimization of frame reconstruction distortion to select key-frames for video summaries. Our proposed approach falls within this particular type of approach. A survey on key-frame extraction methods can be found in the work by Sujatha and Mudenagudi [113].

2.3.2 Video Summarization

These approaches can be categorized into two types [41]: static video abstracts and dynamic video skims. The first category is typically represented by a small collection of key-frames extracted from the video. These key-frames can be used to create tables of contents, storyboards and pictorial video summaries [41]. Note that we do not necessarily need to use real key-frames extracted from the video to create summaries, but we can also generate visualizations describing the content of multiple key-frames. This is particularly useful for representation of dynamic events in static summaries like the work by Nguyen et al. [81] where 3D views are generated to summarize segments of video. Key-frames help to navigate the video in a non-linear way, but most of the dynamics of the content and the audio are lost. Dynamic video skims on the other hand use short segments extracted from the video to create the summaries may be more appealing to the users, and can keep relevant portions of the audio. Note that these two summary types can be combined to create hierarchical summaries [41], where high-level key-frames can be associated with low-level short video segments. Additional information about existing video summarization techniques can be found in the surveys by Truong and Venkatesh [120], and Money and Agius [73].

2.3.3 Evaluation

As noted in the past, ideal key-frames and video summaries tend to be subjective and often their measurement of quality is application dependent [41]. Typical metrics include recall of application-dependent targets extracted in the summary, and video compression ratio where we want to use the smallest possible set of key-frames. Higher recall usually means lower compression ratio by requiring more frames to ensure all targets have been extracted. Choudary et al. [20] uses a recall-based error metric for evaluation of lecture video summaries by identifying missing content by counting connected components for words and lines for graphics. In Chang et al. [16] (1999), the authors proposed a key-frame fidelity measure that is based on a semi-Hausdorff distance.

In some cases, video summarization approaches heavily depend on detecting relevant video segments represented in the summary. This approaches typically measure the quality of split points compared to the ideal partitions of the video. This is particularly necessary in slide-based lecture videos. In the work by Li et al. [54], the evaluation is made in terms of the precision, recall and F-score of the slide transitions detected. In the work by Repp et al. [98], for segmentation of lecture videos based on speech recognition, the computed split points c_i are compared against the ideal set in terms of time difference using the mean absolute error rate. In the AMIGO system [30], the Jaccard index is used to evaluate the correctly detected slide transitions within a ± 3 seconds error margin.

In our work for lecture video summarization presented in Chapter 4, we use the average number of key-frames generated per video as a measurement of video compression, and using videos where all the whiteboard content has been labeled at the connected component, we compute recall as the percentage of successfully extracted connected components, and precision as the percentage of connected components extracted that are part of the targets.

2.3.4 Summarizing Lecture Videos

In our analysis, we focus on three main elements of methods for lecture video summarization and similar applications: binarization/segmentation, content extraction and summarization. Content extraction refers to techniques used to analyze and separate higher level units of whiteboard/blackboard/slide content. Finally, summarization refers to methods used to build compact represents of the video using extracted contents. Some method might use binarization or segmentation before content extraction, while other approaches attempt to extract the contents directly from the raw image before segmenting or binarizing the characters.

We also consider two lecture video content types: handwritten (from paper, whiteboard, chalkboard, etc), and typeset (slides). It is important to note that many approaches are multi-modal and also consider audio and supplementary materials for the lecture. Here, we concentrate on image-based lecture video summarization approaches.

Binarization/Segmentation

For videos or sets of images of handwritten content from whiteboards/chalkboards, traditional binarization methods like the Otsu's [87] have been employed in cases where illumination changes do not represent a major issue like in the work by Wienecke et al. [126] for whiteboard reading. Niblack's binarization method [84] has also been employed for whiteboard reading in the works by Plötz et al. [95] and Vajda et al. [122]. Other thresholding based techniques have also been employed in the works by Liu et al. [63, 64]. Some approaches use text detection and background removal techniques to isolate first the handwritten content regions before thresholding [29, 114, 115].

When the background is not very uniform and simple threshold-based methods fail, more sophisticated segmentation approaches are employed. Color space $L^*a^*b^*$ has been employed for mean-shift segmentation in the work by Comaniciu et al. [21], and k-means segmentation in the works by Chouday and Liu [19, 20, 62] and Lee et al. [53]. In the work by Davila et al., edges detection and morphological operations have been used for whiteboard image binarization [24].

For slide-based lecture videos, the idea of binarizing/segmenting the content is not always an explicit step. Usually, the content is in a format where separating text from graphics is considered trivial after segmenting out the slide from the background.

In our lecture summarization approach described in Section 4.1, we binarize using a background estimation model, and a combination of machine learning with Otsu's [87] binarization method. Random forest are trained using samples with foreground and background patches randomly sampled with bias to prefer pixels near the boundaries of the shapes.

Content Extraction

For whiteboard/chalkboard based lecture videos, different approaches have been used to extract handwritten content. However, some ideas are common between methods. One of these ideas is to divide the input image into a regular grid [24, 29, 63, 64, 122, 126], and then most approaches will try to classify each cell of the grid as handwritten content, background or noise using statistical methods. Some approaches group together cells classified as handwritten content into blocks or text lines that can be used for summarization in the image domain or using OCR methods [95, 122, 126]. Other methods use temporal information available in the video to create these blocks of content using heuristic rules to group together changes as they happen in the image [86, 86]. Some methods employ local features extracted to classify patches in the image as text or background. In the works by Tang and Kender [114, 115], SVMs are trained using features extracted from the edge image. In the work by Banerjee et al. [9], SIFT [65] features extracted on a regular dense grid are employed for patch-based classification of pixels as text/background using MLPs.

A common factor affecting the quality of the extracted content is low resolution. Some approaches like the work by Tang and Kender [114, 115] have employed super resolution techniques taking advantage of the temporal information available in the video domain. Another common factor affecting results and readability of the raw material is poor contrast. Some methods employ contrast enhancing techniques to deal with this issue [29, 95, 122].

To take advantage of the particular constraints of the lecture video domain is usually helpful. For this reason, multiple approaches resort to explicit recognition of elements or events that are expected to appear. Multiple approaches have used explicit speaker modeling to improve the precision of their methods by avoiding extraction of occluded content [20, 24, 29, 43, 53, 54, 86, 107]. Detecting erasing events in whiteboard/chalkboard based videos is also helpful for video segmentation, and usually the whiteboard will be full of content that can be used to summarize the segment right before these erasing events happen [20, 24, 43]. Some methods only extract content from frames with little motion assuming that this means no particular events like erasing or writing are taking place and that the speaker is probably not currently in the view of the camera [1, 24, 114, 115]. Also, not all methods need to detect the whiteboard, chalkboard or slide area in the image explicitly before doing content extraction, but some existing methods do it for different reasons including increased precision of content extraction [24] and the ability to correct camera view distortions for cleaner content extraction [54].

For slide based videos, detecting transitions between slides is analogous to detecting erasing events in whiteboard/chalkboard based videos. Once slide transitions have been detected, key-frames where they are more readable can be chosen for each slide and then OCR techniques can extract the typeset text they contain. If supplementary materials are available, they can be used to improve or constrain the slide detection and content recognition results [30, 107]. A full-optimization framework based on local feature tracking is used in the work by Li et al. [54] for detection of slide location and transition. Sometimes video production can include video segments where the slides, whiteboard or chalkboard are not visible. Such segments might be focusing on the audience, the speaker or any other element in the room. Detecting such segments is a challenge but also a great help in obtaining clean sections of content out of the video. In the work by Adcock et al. [1], a SVM is trained for classification of slide/non-slide key-frames on videos. In the work by Li et al. [54], sharp changes in local feature tracking are used to detect and exclude these segments.

Content Summarization

The units used to create summaries of lecture videos will depend on the particular target of the application. For whiteboard reading approaches, the final summary is given by text lines extracted and recognized text from the video [43, 95, 114, 115, 122, 126]. Region-based content extraction methods use extracted regions of content to represent the input video [24, 86]. Images coming from different camera views of the content can be mosaicked into single panoramic images [20, 53] or into large virtual slides [64].

Full key-frames can be used to summarize the video as well. Some methods try real-time selection of key-frames for lecture video summarization typically using rules based on motion and/or content change detection [19, 20, 29, 53, 63, 114, 115] where frames with low motion and large content changes from the previously selected key-frame are preferred. In particular, some methods try to identify peaks in a function that sums all chalk/ink pixels per frame for key-frame selection [20, 53]. For slide-based videos, images of the detected slides are also typically used as video summaries [30, 54]. In the work by Eberts et al. [30], local features are used for indexing of the graphic content embedded in the slides. Yadav et al. [129] uses C-NN to detect and index anchor elements (figures, tables, equations, proofs, etc) in images for indexing and summarization of slides.

Our method described in Chapter 4 uses full key-frames for summarization. However, one important difference is that these key-frames are generated from a spatio-temporal structure that contains very detailed information about the video allowing for very dynamic summaries. Using these key-frames and the original spatio-temporal structure, we have created our own navigation tools that let the user click on particular connected components of the key-frames to jump to the video segment where that particular element was written on the whiteboard[5].

2.4 Mathematical Symbol Recognition

Math symbol recognition is a required element of our proposed approach for image-based formula retrieval described in Chapter 6. We also proposed our own set of features for on-line math symbol recognition along with a synthetic data generation method and the required adaptations to use this approach with off-line math symbols in Chapter 5. In this section we describe some other existing approaches for math symbol recognition.

Different approaches have been used before in the field of math recognition and retrieval as documented by Zanibbi and Blostein [137]. The CROHME competitions [74, 75, 76, 77, 78, 79] provide a good comparison between different approaches for math expression recognition on a set of standardized tasks. For the scope of this section, we are only interested in the specific problem of isolated on-line math symbol recognition.

One of the most important elements of a math symbol classifier is the set of features used to describe each symbol. In the survey by Yang et al. [130] different feature extraction techniques for shape description have been identified. While some feature extraction techniques based on timing or drawing information have been used before for on-line symbol recognition [5, 6, 34, 39, 118], techniques based on shape description might be more robust since a single shape can be drawn in several ways. In the work by Delaye et al. [28] a set of 49 features that describe both the drawing process and the shape have been proposed as a baseline for different on-line symbol recognition datasets.

In addition to strong features, good training data is also important. For some applications, synthetic training samples can be generated to improve the quality of a given training set. In the current application, it is possible to apply a data generation model to create synthetic samples from the existent. In the work by Simard et al. [108], an elastic distortion model for data generation is applied to off-line character recognition. Also, the Sigma-lognormal model of the kinematic theory of rapid human movement presented in the work by Plamondon et al. [94] has been used to produce very realistic synthetic on-line signatures and handwritten gestures.

Nevertheless, it has to be considered that on a recognition problem there might be different styles for a single class [101]. For handwritten symbols, these styles go beyond small differences in slant and thickness to the point of having completely different shapes that still belong to the same class like for example the lowercase z character written in cursive style. If a given writing style is absent from the training set, then it will be very difficult or even impossible for a classifier to learn to correctly identify symbols using that style.

2.5 Summary

In this chapter we have covered related works from different areas that are relevant to our own work. We covered four main areas of research: mathematical information retrieval, content-based Image retrieval, lecture video summarization, and mathematical symbol recognition.

In Section 2.1, we discussed existing methods for formula retrieval in two modalities: symbolic and image-based. Math in published documents is found in different formats including LaTeX, Presentation MathML and Content MathML. We discussed multiple text-based, tree-based, and

[5]https://www.youtube.com/watch?v=rsGRAsjTN4Y

spectral representations that MIR systems use to store these math expressions in their indices. Different model have been proposed to rank documents that contain matches for query formulas mainly based on similarity metrics. Some systems allow the users to find additional non-exact matches for their queries through partial matching, unification and wildcard matching. Finally, while most of these processes are well defined for symbolic formula retrieval, there are less standard approaches proposed for recognition-free, image-based formula retrieval. Additional details about approaches for mathematical information retrieval can be found in the works of Zanibbi and Blostein [137], and Guidi and Coen [36].

Our models uses a combination of different elements that have produced promising results on each of the main aspects of a math information retrieval system. In the symbolic domain, we have adopted tree-based representations built from presentation and content MathML for symbolic formula retrieval. Our inverted index is based on tuples extracted from these tree-based representations, and we have considered different metrics to quantify notions of similarity like the ones discusses in Section 2.1.1.

In Section 2.2, different CBIR image retrieval approaches have been presented. The BoVW approach is still one of the best frameworks for scalable retrieval of images. Certain elements like the specific local descriptors, clustering methods, and spatial verification models can greatly affect the final performance of BoVW approaches. Other approaches for CBIR work with images of handwritten sketches which different properties compared to natural scene imagery. Our method for image-based formula retrieval follows the direction of some approaches discussed before by employing graph-based representation for global structural matching, but replaces local descriptors by actual symbols recognized from Connected Compontents on the binary images.

In Section 2.3, we have discussed some general ideas about video summarization including concepts of key-frame extraction, types of video summaries and evaluation metrics. We described the three main challenges of summarizing lecture videos: Binarization/segmentation, content extraction, and generation of summaries. We also discussed how some of the existing approaches in the field have dealt with each of these challenges. In our own work, we have taken advantage of the domain restrictions given and we have proposed our own method for lecture video summarization (described in Chapter 4) that we use as a preprocessing step in order to achieve retrieval of mathematical notation from lecture videos.

Finally, in Section 2.4, we discussed existing approaches for math symbol recognition. We also covered some of the existing work in synthetic data generation for improvement of symbol recognition in general.

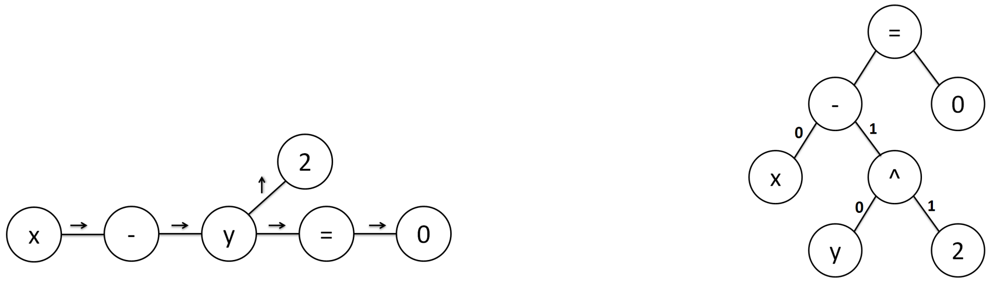
Chapter 3

Retrieval of Mathematical Notation in Documents

In Chapter 2, we discussed existing approaches for symbolic math information retrieval. In this chapter, we present our model for retrieval of formulas in the symbolic domain. The work presented here describes the Tangent-3, Tangent-3.1 and Tangent-S formula retrieval models which are a continuations to previously existing models for formula retrieval developed by the Document and Pattern Recognition Lab at Rochester Institute of Technology. Table 3.1 summarizes all versions of the Tangent system so far.

In Tangent-S, we use both appearance-based representations of mathematical expressions (through the arrangement of symbols on writing lines) in *Symbol Layout Tree (SLT)* or semantics in *Operator Tree (OPT)* [137]. LaTeXand Presentation MathML are examples of SLT representations while Content MathML is an example of OPT representation. Figure 3.1 shows the corresponding SLT and OPT representations for the expression $x - y^2 = 0$.

Both representations have been used for query-by-expression. Many researchers in *Mathematical Information Retrieval (MIR)* assume OPTs provide better formula retrieval results than SLTs, but each has limitations for retrieval. For SLTs, mathematical notation can change meaning based on context - a symbol may be an operator in one context, and a variable in another, for example. In contrast, well-formed OPTs are mathematically unambiguous. Online, most users write math expressions using SLT representations (e.g., LaTeX). SLTs can be converted to OPTs using parsers, but semantics are often undefined or ambiguous, producing errors [36].



(a) Symbol Layout Tree (b) Operator Tree

Figure 3.1: Tree representations for $x - y^2 = 0$

Table 3.1: Summary of Tangent Formula Retrieval models

Model	Domain	Formula Repr.	Description	Authors	Details
Tangent	Symbolic	SLT	First version, indexed formulas based on symbol pairs. Used horizontal and vertical displacements to model relationships between symbols	D. Stalnaker R. Zanibbi	[112]
Tangent-2	Symbolic	SLT	Improved SLT representation to incorporate matrix support. Incorporated support for queries mixing keywords and math expressions.	N. Pattaniyil R. Zanibbi	[88]
Tangent-3	Symbolic	SLT	Improved SLT representation. New representation for grouped elements including matrices. Symbol types incorparted and seven edge types defined. Symbol pair model now uses paths from the SLT to represent relationships between ancestor/descendant symbols. Two-layer retrieval model introduced with new faster core-engine in C++ and detailed matching algorithms for candidate re-ranking based on the MSS similarity metric. Wildcards match single elements only, and unification between symbols of the same type is also introduced.	K. Davila R. Zanibbi A. Kane F. Tompa	[138], Chapter 3
Tangent-3.1	Symbolic	SLT	Two new edge types incorporated (over and under) for modeling of expressions that combined accents with superscripts. Wildcard expansion added allowing wildcards to match entire subtrees and expand horizontally across baselines for SLTs. New scoring metrics and tie-breakers.	K. Davila R. Zanibbi A. Kane F. Tompa	[27] Chapter 3
Tangent-S	Symbolic	SLT, OPT	Introduced support for operator trees. New matching algorithms specially designed for greedy matching of children for commutative operators. Symbol type system refined and new symbol subtypes incorporated for more accurate unification results. New optional third layer for retrieval using linear regression, and improved formula retrieval result using both SLT and OPT at the same time.	K. Davila R. Zanibbi	[26] Chapter 3
Tangent-V	Image-Based	LOS-Graph	Novel image-based formula retrieval model. Based on partial symbol recognition results. Considers multiple labels per symbol pair and a 3D-unit vector system for modeling of relationships between symbols. Structure is represented using refined LOS-graphs.	K. Davila R. Zanibbi	Chapter 6

In the most general sense, query-by-expression involves tree matching for a class of SLTs or OPTs. Open problems include defining appropriate similarity metrics, identifying what primitives to use for representing formula content, how to index and retrieve these primitives, and finally how to handle trade-offs between recall and both retrieval time and index size. In general, retrieval becomes more difficult when the best matches are less similar to the query formula (e.g., due to large expressions in the corpus with subexpressions similar to one or more parts of the query), and

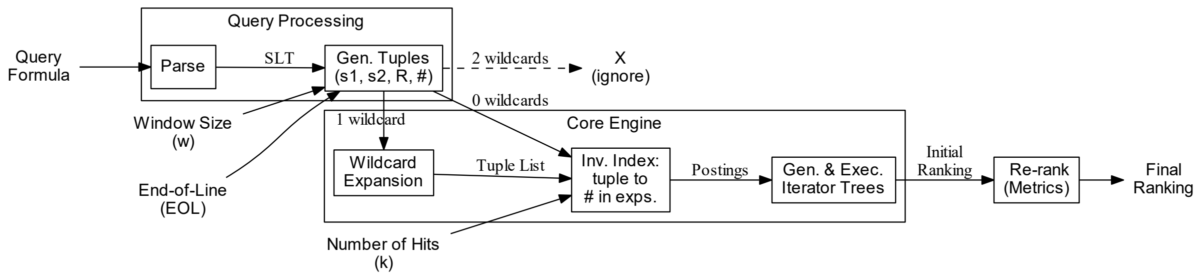


Figure 3.2: Tangent-3 Formula Retrieval Model. System parameters include maximum symbol pair distance (window size w), how end-of-line symbols are indexed (EOL), and the number of hits to return (k). Different similarity metrics can be used in the final re-raking process.

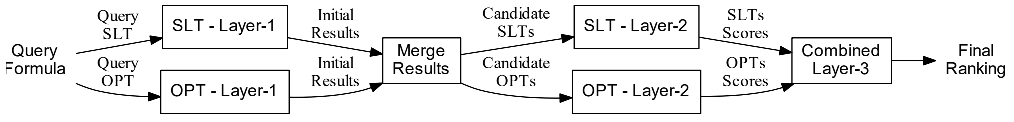


Figure 3.3: Tangent-S Combining Formula Representations. Two parallel indices are queried, one for each representation. Initial results from each index are merged and matching is applied over the complete set of candidates using both representations. Final score is obtained using Linear Regression over the complete set of similarity scores from both representations combined.

when wildcards are included in the query [46].

Our model Tangent-S[1] [26, 27, 138] uses a three-stage model for formula retrieval. First, given a particular representation, SLT or OPT, top-k candidates are identified using symbol pairs in representation trees as ‘words’ (See Figure 3.2). Then, in the second stage, the top-k candidates are re-ranked after aligning query and candidate trees while considering unification and wildcard matching. Candidates are re-ranked using the harmonic mean of symbol and relationship recall (the *Maximum Subtree Similarity*) and two tie-breakers: symbol precision after unification, and symbol recall without unification. Finally, in the third stage linear regression is used to combine all similarity scores into a single score predicting relevance of each match.

It is also possible to use two parallel formula indices, one for SLTs and the other for OPTs, and execute the initial retrieval step in parallel (See Figure 3.3). Then, initial candidates from each index are merged into a list of unique candidates. Note that some unique math expressions will be matched by both representation, but many of them will be matched by one of them only. The merge step retrieves the missing representation for these candidates, and then step 2 is applied for each candidate using both representations in parallel again. Finally, the similarity scores from both representations are combined in the final linear regression step to give each candidate a final relevance score.

Tangent-3 [138] and Tangent-3.1 [27] only use the first two layers of the retrieval process. Tangent-3.1, considers a more extensive set of tie-breakers including statistics of matching wildcards and left-to-right location of the matching components [27] (see Table 3.1).

A simple extension of the model is also considered to deal with queries containing multiple

[1]`https://cs.rit.edu/~dprl/Software.html\#tangent`

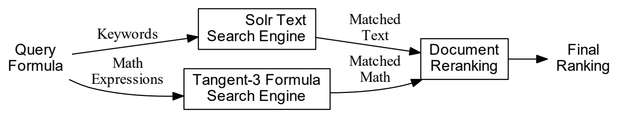


Figure 3.4: Tangent-S Full Retrieval Model. Text and math expressions are indexed and queried separately, and partial results are combined to produce a final document ranking.

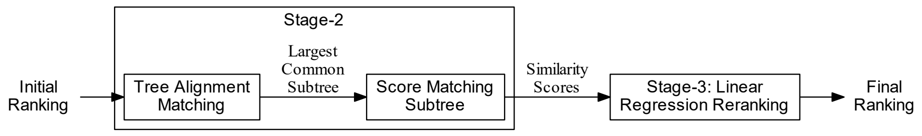


Figure 3.5: Tangent-S Formula Re-ranking Stages. Initial candidates are received from Stage 1, the core- engine, and a detailed matching process is applied in Stage 2. The largest common subtrees found between the query and each candidate are then scored using multiple similarity metrics. Finally, on Stage 3, these similarity scores are combined into a single final metrics used to produce a final Ranking of formulas for a given query.

mathematical expressions and/or text. Multiple indices are used, the first for text using Solr[2], and the remaining for math expressions using SLTs and/or OPTs. For each query, the system processes the keywords and math expressions separately to find relevant documents for each, assigning relevance scores to each document where a match is found (see Figure 3.4). Once text and math expression search are complete, a combination step is required to produce a final ranking, where documents that contain matches for both keywords and math expressions will be ranked higher than documents containing matches for just text or math.

Most queries in the NTCIR-12 MathIR task [136] contain multiple keywords and at least one math expression. Note that in our work we use the same keyword indexing and matching procedure from Tangent-2 [88], and we use a linear combination of relevance scores for matched text and math expressions to rank documents for these queries. We also experimented with different ways of setting the linear combination weights.

3.1 Formula Representation

The Tangent-S formula model uses both SLT and OPT to represent formulae (see Figure 3.1). To define and constrain the behavior of matching and unification algorithms, we assign each symbol a type. Edges between symbols are labeled by their order in OPTs, or by the visual location of a child symbol with respect to its parent (e.g., for superscript relationships) in SLTs (see Fig. 3.1). In this section, we first describe the symbol types used by both representations, and then we describe the particularities of each formula representation.

[2]http://lucene.apache.org/solr/

3.1.1 Node Types

Nodes in an SLT represent individual symbols and visually explicit aggregates, such as fractions, matrices, function arguments, and parenthesized expressions (see Figure 3.6). For OPT, internal nodes will always represent operations while operands will be leaves of the tree, making it easier to distinguish between identifiers that represent function names and those representing variables.

Every node has a label, and a node's type (*number*, *variable*, *operator*, etc.) is reflected in its label. If a node's label includes an exclamation mark (e.g., V!), the type is the label prefix up to the (first) exclamation mark. Node labels starting with an asterisk (*) have type *wildcard*. For SLTs, node labels without exclamation marks have type *operator*. More specifically, the tree node types that we are considering are:

- Numeric constants (N!)
- Identifiers such as variables (V!)
- Text fragments, such as *lim*, *otherwise*, and *such that* (T!t)
- Function names (F!)
- container objects: matrices, vectors, sets, lists, tabular structures, and parenthesized expressions (M!)
- explicitly specified whitespace (W!)
- Wildcard symbols (*w)
- Commutative or Unorderer operators (U!)
- Non-commutative or Ordered operators (O!)
- Parsing Error (E!)

In Tangent-3 [138], fractions and radicals were represented using their own node types (F! and R! respectively). As we extended our model to consider more spatial relationships [27] and differentiate between commutative and non-commutative operators [26], these two elements became part of the Orderer Operator subtype (O!) because both are non-commutative operators for both SLTs and OPTs.

Real data often provides strong clues for symbol types, but in some cases symbol type can be hard to infer without context, leading to incorrect symbol types and invalid unifications.

3.1.2 Symbol Layout Tree (SLT)

This representation is built around writing lines (*baselines*), generally leading to deep trees with few branches. The children of a node in this representation are assigned to a spatial relationship class (edge label). A previous version of the SLT model used in Tangent-2 [88] based its encoding on a two-dimensional interpretation of formulas on a page, expressing symbol positions in terms of horizontal and vertical offsets. The revised SLT representation used in Tangent-3 and Tangent-S provides greater consistency and expressivity in representing relationships between symbols because of the new edge labels that are much more specific than the original offset system.

Matrices and other groupings are an integral part in our proposed representation, capable of containing formula subtrees rather than being simple auxiliary relational structures. Because of their visual similarity, all tabular structures, including matrices, binomial coefficients, and piecewise defined functions are encoded using the matrix indicator M! regardless of their interpretation (for example, as function arguments vs. parenthesized matrices). If a matrix-like structure is surrounded by fence characters, then those symbols are indicated after the exclamation mark. Then, the number of rows and the number of columns separated by an x are also added to the M! indicator. For example, M!2x3 represents a 2x3 table with no surrounding delimiters and M!()1x5 represents a 1x5 (1 row by 5 column) table surrounded by parentheses. Importantly, *all* parenthesized subexpressions are treated as if they were 1x1 matrices surrounded by parentheses, and, in particular, the arguments for any n-ary function are represented as a 1xn matrix surrounded by parentheses.

Spatial Relationships. Labeled edges in the SLT capture the spatial relationships between objects represented by the nodes. With respect to a given object O, nine axes reflect the following relationships:

1. **next** ($\rightarrow$) references the adjacent object that appears to the right of O and on the same line
2. **within** ($\boxed{\cdot}$) references the radicand if O is a root or the first element appearing in row-major order in O if it is a structure represented by M!
3. **element** ($\multimap$) references the next element appearing after O in row-major order inside a structure represented by M!
4. **above** ($\uparrow$) references the leftmost object on a higher line starting at the position above O (e.g., superscript, or index for a radical)
5. **below** ($\downarrow$) references the leftmost object on a lower line starting at the position below O (e.g., subscript)
6. **Over** ($\Uparrow$) symbol, references an element directly over O, like a fraction numerator or an accent
7. **Under** ($\Downarrow$) symbol, references an element directly under O, like a fraction denominator, or underline
8. **pre-above** ($\nwarrow$) references the leftmost object of a prescripted superscript of O
9. **pre-below** ($\swarrow$) references the leftmost object of a prescripted subscript of O

Creating SLTs. SLTs can be created in linear time from Presentational MathML by a recursive descent parser. For other input formats, we assume that converters such as LaTeXML[3] exist to produce Presentational MathML.

In most circumstances, whitespace is not represented in an SLT. As a result, although unicode whitespace and related characters, such as "invisible times" (`U+2062`), occasionally appear as operators in Presentational MathML expressions, they are all ignored for the purpose of matching expressions in Tangent-S.

3.1.3 Operator Trees (OPT)

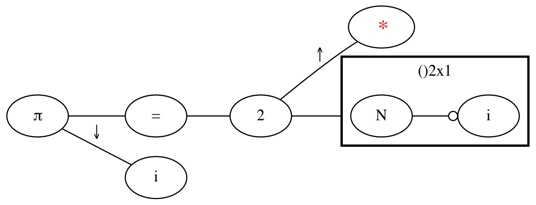
This representation is built around the hierarchy of operators in a formula, generally resulting in shallow trees with many branches. We distinguish between commutative operators (e.g., '+') and non-commutative operators (e.g., '-'). We ignore the order of children for commutative operators. In Figure 3.1.b, all edges to children of the equals sign have the same label. Note that fence symbols in matrix-like elements are not used for operator trees as they are not even represented by Content MathML.

For certain operators like roots, summations, and integrals, the order of the children determines their meaning to the operator. For example, the first child of a root is always the operand of the root, and the second child is the order of the root (squared, cubic, etc). If the operand is not specified but the default is known (roots are squared roots by default) then the default operand is added automatically. In other cases, where no default exists, a dummy operand node is added.

A subtype system is added to improve the accuracy of unification in cases where enough information is available during parsing. For example, we distinguish between sets, lists, vectors and other matrix-like elements thanks to the fact that these are explicitly declare in the source Content

[3]`http://dlmf.nist.gov/LaTeXML/`

$$\pi_i = 2^{*}\binom{N}{i}$$



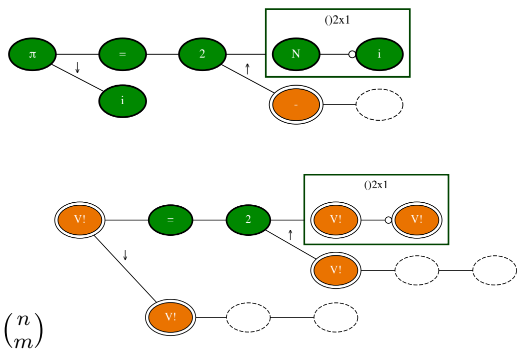
(a) Query Formula and Symbol Layout Tree (SLT)

Sym-1	Sym-2	Path	Count
V!π	**V!**i	$\downarrow$	1
V!π	=	$\rightarrow$	1
=	**N!**2	$\rightarrow$	1
N!2	*x0	$\uparrow$	1
N!2	**M!**()2x1	$\rightarrow$	1
M!()2x1	**V!**N	$\boxed{\cdot}$	1
V!N	**V!**i	$\multimap$	1
V!N	!0	$\rightarrow$	1
V!i	!0	$\rightarrow$	2
M!()2x1	!0	$\rightarrow$	1
*x0	!0	$\rightarrow$	1
V!π	**N!**2	$\rightarrow\rightarrow$	1
=	*x0	$\rightarrow\uparrow$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
V!π	**V!**i	$\rightarrow\rightarrow\rightarrow\boxed{\cdot}\multimap$	1

(b) Tuples for Query Formula Symbol Pairs including Counts and Relationships

$$\pi_i = 2^{-N}\binom{N}{i}$$

$$E_{m,n} = 2^{n-m}\binom{n}{m}$$



(c) Example Search Hits. Green: exact matches, Orange: unified matches, Dashed Nodes: unmatched

Figure 3.6: Query Formula with Corresponding SLT. The query has one wildcard, and a tabular structure.

MathML.

We also specify the arguments of functions as children of a function node instead of using a group of elements like it is done for Symbol Layout Trees. While functions are not explicitly defined in Content MathML, we distinguish them when an identifier is the first child of the Apply tag. For example if f and x appear within the same Apply tag and f is the first element, then we will assume f is a function, because the Apply tag in Content MathML requires the operator to be specified as the first child.

3.2 Pair-based Index Model

An effective formula search engine must be able to find formulae that contain a query formula, appear within a query formula, or are in many ways only similar to a query formula. Thus high-quality search engines create indexes based on selected *features* of formulae found in a corpus and match queries based on those features. Previous versions of Tangent showed that pairs of symbols together with their relative inter-symbol distances in two dimensions are effective features to use [88], and we improve on this approach.

For a given tree representation, Tangent produces a set of tuples, each of which encodes the relationship between a pair of symbols occurring on some path from the root to a leaf and the number of times each such label-label-path triple occurs. Given two nodes on such a path, we define the relative path between the nodes by the sequence of edge labels traversed from the ancestor node to the descendant.

Tuple Representation for SLTs. As described above, a node in an SLT can have up to nine labeled outgoing edges (with no edge label repeating for any node) corresponding to the nine defined axes. SLT tuples use these sequences of labeled edges to define paths that represent relative layout locations between symbols. For example, the tuple ($V!x$,$N!2$,$\rightarrow\uparrow\rightarrow\rightarrow$,3) indicates that the corresponding formula includes three instances of a node representing the number 2 that appears "to the right, then above, and then twice to the right" with respect to some node representing variable x; for example, such a formula might include ...xy^{z+2}....

Tuple Representation for OPT. Edge labels in operator trees have different meanings based on the label of each internal node. In this sense, we distinguish between commutative (Ordered O!) and non-commutative (Unordered U!) operators. Ordered operators will assign edge labels to their children based on their relative order, while unordered operators assign the same edge label to all children. Using different labels for each kind of operator we are able to generate tuples that will be shared by different non-identical but equivalent trees. For example, the expressions $1 = y^2 + x$ and $x + y^2 = 1$ will produce the same exact set of tuples and can be matched as identical expressions despite having a different order of operands for the commutative operators $=$ and $+$.

Tuple Generation Parameters. As seen in Figure 3.2, there are two parameters that control symbol pair tuple generation: the window size (w), and how symbols at the end of writing lines are included in the index (EOL).

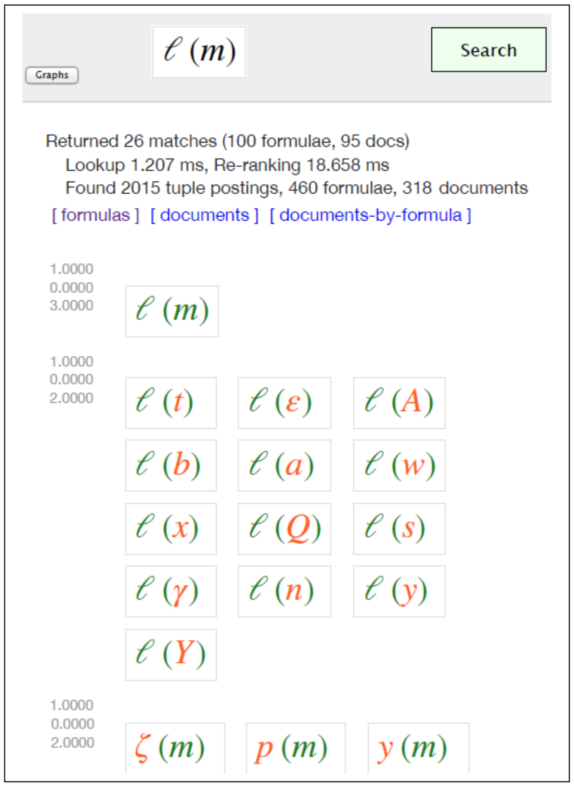
To save both space and time, and following the practice of searching via n-grams [127], Tangent-3 extends the approach used in Tangent-2 by storing only those tuples for which the distance between symbols (measured by the number of edges separating them) is less than or equal to a specified window size w. For example, the tuple ($V!x$,$N!2$,$\rightarrow\uparrow\rightarrow\rightarrow$,3) will be included in the index only if $w \geq 4$.

In addition to symbol pairs, end-of-line information can be captured by introducing special tuples of the form (*symbol*, !0, $\rightarrow$, *count*). Including these is likely to improve retrieval, particularly for very small expressions. However, wildcards as EOL symbols have very large wildcard expansions, increasing retrieval time. To alleviate this problem, we examine adding EOL symbols for small expressions with height two or less (*Sm-EOL*), adding all EOL symbols (*EOL*), and omitting EOL symbols (*No-EOL*).

3.3 Formula Retrieval Model

The Tangent-3 formula search engine [138] employs a two-stage cascading search [124] for fast retrieval and intuitive rankings (see Figure 3.2). Queries can be parsed into either Symbol Layout Trees or Operator Trees (see Figure 3.1), which are then traversed from the root, generating tuples of the form $(s_1, s_2, R, \#)$ with ancestor symbol s_1, descendant symbol s_2, edge label sequence R from s_1 to s_2, and a count ($\#$). Two parameters control the maximum path length between symbols in tuples (the window size, w) and whether to include tuples for symbols at the end of writing lines (EOL).

Applying detailed similarity metrics can be prohibitively expensive. For this reason, a three-layer retrieval process is used. After parsing, the first retrieval stage (the *core engine*) ranks a given number of expressions k by matching query tuples, using an inverted index mapping symbol



Returned 26 matches (100 formulae, 95 docs)
Lookup 1.207 ms, Re-ranking 18.658 ms
Found 2015 tuple postings, 460 formulae, 318 documents
[formulas] [documents] [documents-by-formula]

1.0000
0.0000
3.0000
$\ell\,(m)$

1.0000
0.0000
2.0000
$\ell\,(t)$ $\ell\,(\varepsilon)$ $\ell\,(A)$
$\ell\,(b)$ $\ell\,(a)$ $\ell\,(w)$
$\ell\,(x)$ $\ell\,(Q)$ $\ell\,(s)$
$\ell\,(\gamma)$ $\ell\,(n)$ $\ell\,(y)$
$\ell\,(Y)$

1.0000
0.0000
2.0000
$\zeta\,(m)$ $p\,(m)$ $y\,(m)$

Figure 3.7: Tangent-S Search Results Page Example. Top-3 matching groups for the query $l(m)$. Top match is a perfect match of the query. The second group of matches show unifications for the variable name used. The last group shows unification for the function name using the same variable. The second and the third group are actually tied in their similarity scores, and the order in which they appear will depend on the order in which they were indexed. Note that results are truncated for space.

pair relationships to expressions and counts (see Section 3.3.1). Tuples with one wildcard are expanded, but tuples with two wildcards are ignored for efficiency. Iterator trees are used to process postings quickly. The initial ranking weighs matched vs. unmatched symbol pairs in the query and candidates. The second (*re-ranking*) stage finds an approximate best matching subtree and re-scores candidates for the query using a detailed similarity metric (see Section 3.3.2).

A third stage is added in the Tangent-S formula retrieval model [26] (see Section 3.3.3). This stage uses a linear combination of similarity scores computed from the second layer to produce a final re-ranking of the top-k candidates. Rather than use a learning-to-rank technique [61], a simpler model was chosen for better understanding of the relevance of each similarity factor.

Illustration. Table 3.2 shows queries processed using the first two stages of our method. For query 1, using MSS for re-ranking produces top-5 hits matching the query formula exactly after unifying identifiers (before re-ranking, only the top-3 match). For query 2, the numbered wildcards *1* and *2* should be identical when repeated. Before re-ranking only one hit matches the query exactly (rank 4), but after re-ranking the top-3 are exact matches for the query, and the remaining two hits are strong partial matches.

Queries containing keywords. In this work, we use a simple model where keywords and math expressions in a query are processed separately and results are then combined to produce a

Table 3.2: Tangent-S Wildcard Matching Examples. Top-5 Results for two queries containing wildcards. Asterisks represent wildcards (e.g., $*$ or $*1*$). Query-2 uses bounded wildcards

	Query 1: $f_*(z) = z^2 + c$	
	Initial Ranking	**Re-ranked**
1.	$f_c(z) = z^2 + c$	$f_c(z) = z^2 + c$
2.	$f_c(z) = z^2 + c.$	$\mathbf{P}_c(z) = z^2 + c$
3.	$f(z) = z^2 + c$	$f_c(\mathbf{x}) = \mathbf{x}^2 + c$
4.	$f_0(z) = z^2$	$f_c(z) = z^2 + c.$
5.	$f_c(z) = z * z + c$	$f(z) = z^2 + c$
	Query 2: $\sum_{*2*}^{*1*} * = \sum_{*2*}^{*1*} *$	
	Initial Ranking	**Re-ranked**
1.	$E = \sum_i^N E_i$	$\sum_{i=1}^d a_i = \sum_{i=1}^d b_i$
2.	$G_{net} = \sum_i \sum_{i=1}^N$	$\sum_{i=1}^N d_i = \sum_{i=1}^N \lambda_i.$
3.	$\sum_i^{N_1} p_i = \sum_j^{N_2} p_j$	$\sum_{n=0}^\infty a_{\sigma(n)} = \sum_{n=0}^\infty a_n.$
4.	$\sum_{i=1}^n x_i k_i = \sum_{i=1}^n x_i$	$\sum_i^{N_1} p_i = \sum_j^{N_2} p_j$
5.	$= \sum_{k=1}^n a_k$	$\sum_{n=0}^\infty a_n = \sum_{n\in N} a_n.$

final document ranking as illustrated in Figure 3.4 and described in Section 3.3.5.

3.3.1 Layer 1: Initial Candidate Selection

The first layer of the retrieval model receives as input a set of symbol pair tuples extracted from the query, and quickly find potential matches from the index. The indexing sub-program, which we call here the **Core Engine**, implements the required data structures that make such process very fast allowing to scale the system.

Core Engine

The core engine is the first retrieval stage in Tangent-S, quickly finding the top-k highly relevant matches for a formula query. The engine uses an approximation of the Dice coefficient of recall and precision of matching symbol pairs between the query and each candidate. The engine ranks candidates based on this score, and returns the top-k unique formulae along with a list of document locations for each formula.

Since runtime performance is a high priority, the core engine of Tangent-S uses a customized inverted index data structure implemented in C++. In addition, the engine evaluates only a subset of the query language functionality to allow the use of a fast and simple ranking algorithm that can still find a good set of candidate results.

The input to the indexer is a set of document names and the extracted mathematical formulae found in each document, $\{document, formula^+\}^*$, and the input to the search engine is a single

Dictionaries

D_f:	formula $\rightarrow$ formID
D_t:	tuple $\rightarrow$ tupleID
D_d:	document $\rightarrow$ docID
D_w:	wildcardtuple $\rightarrow$ wildcardtupleID

Postings lists

P_t:	tupleID $\rightarrow$ (formID, count)$^+$
P_f:	formID $\rightarrow$ (docID, position)$^+$
P_w:	wildcardtupleID $\rightarrow$ (tupleID)$^+$

Figure 3.8: Primary data structures used in the core search engine.

query formula. Each formula is converted to a set of tuples (see Section 3.2) that serve as index and search "terms."

Parameters. In addition to the window size (w) and End-of-Line (EOL) parameters used in tuple generation (see Section 4), the core engine has a parameter for the number of formulae k to return for each query. The runtime efficiency and ranking effectiveness of configurations using various settings of the first two parameters with $k = 100$ are examined in Section 3.4.

Index Data Structures. At index time, an inverted index is built over the given document-formula-tuple relationships, using two main data structures: *dictionaries (D_*)* convert objects (such as strings or tuples) into a compact 0-based range of internal identifiers (integers) and *postings lists (P_*)* are lists of integer tuples ordered by the first integer in the tuple. The data structures listed in Figure 3.8 can be combined to produce ease of storage, and fast access speeds.

The index includes postings lists P_t that map each tuple to all formulae containing that tuple. A query can thus be implemented by combining the corresponding tuples' postings lists using an OR operator. We store these postings lists as ordered lists of formula identifiers (integers), so that the lists can be easily combined using a merge algorithm. Dictionary D_f defines a consistent assignment of a formula to its identifier, and another dictionary D_t is used for tuples, thus saving both space and time in the engine.

In order to return document information for query results, the engine stores postings lists P_f mapping each formula identifier to the identifiers of the documents containing those formulae, along with their positions in documents. Dictionary D_d is used for document names.

Wildcards. The core engine supports limited wildcard functionality. As illustrated in Figure 3.4, query tuples containing a single wildcard are implemented as iterator expansions. The engine stores postings lists P_w that map each wildcard tuple to the set of tuple identifiers that match. Assigning tuple identifiers using a dictionary D_w again gives some compression benefits. Implementing even this restricted wildcard functionality can be expensive, since the iterator expansion can be quite large.[4]

Searching. Query processing follows the architecture shown in Figure 3.2. First, the query is parsed into either an SLT or OPT, and tuples are extracted. Then wildcard tuples are expanded, the associated postings lists for each tuple are found, iterators over these lists are created, and an iterator tree that implements the query is formed. Next, the iterator tree is advanced along formula identifiers in order, the scores are calculated, and the top-k formulae are stored in a heap. During this process, non-wildcard iterators are advanced first so that wildcard iterators only match

[4]The engine does not try to enforce wildcard variable agreement between tuples (wildcard joins), and it ignores multi-wildcard tuples. An initial implementation handling multi-wildcard tuples and wildcard joins was found to be approximately one hundred times slower for a small dataset.

unallocated tuples. As optimizations, iterators may skip over some formulae based on thresholds and max-score calculations (*see below*). After the iterators are finished, matching formulae and scores are returned along with the associated document names.

The engine uses Dice's coefficient over tuples as a simple ranking algorithm, counting the number of tuples that overlap between the query and a candidate formula using the query iterators. The engine also stores the tuple count for each formula (the size of the formula) in an array A_s and uses these values in the ranking calculation:

- A_s: formID $\rightarrow$ tuplecount

Since wildcards can often match multiple tuples in a query and overlap with other wildcards, there could be multiple ways to count the tuples that overlap. The engine implements a greedy counting approach by simply assigning the matches for tuples when each of the iterators is advanced.

Optimizations. Even using simple dictionary and postings list implementations (i.e., std::maps and 32-bit arrays), the engine's data structures are small enough to be run in memory for the datasets being examined, so we do not examine additional techniques to compress these data structures here. Nevertheless, query processing might still be slow, even though the data structures are in memory, the ranking algorithm is fast, and the use of a dictionary avoids repeated processing of duplicate formulae. As a result, various techniques are employed to improve query execution time:

O_1: Avoid processing all postings by allowing skipping in query iterators. This functionality is implemented using doubling (galloping) search [11].

O_2: Skip formulae based on size thresholds. We use the current top-k candidate list to define a minimum score that defines minimum and maximum tuple size thresholds from the definition of Dice's coefficient. We also improve on the effectiveness of these thresholds by reordering formula identifiers: sort the formulae by size, split into quartiles $\{q_1, q_2, q_3, q_4\}$, and then reorder $\{q_2, reverse(q_1), q_3, q_4\}$.

O_3: Avoid formulae that match only wildcard tuples when the score threshold allows. This is similar to portions of the max-score optimization [121], only at a coarser granularity.

O_4: Avoid processing all wildcard tuple expansions. If a tuple is matched to a wildcard for the next formula, do not process the remaining iterators for this wildcard.

O_5: Process iterators for large postings lists first. Evaluate the binary operator tree left-first and order tree operators descending by size when possible.

3.3.2 Layer 2: Structural Match Scoring

After selecting an initial set of candidates using the Core engine ranking algorithm, the next step is to apply a detailed matching algorithm to find the largest common subtree between the query and each candidate (**Matching**), and then use this largest subtree to compute multiple similarity metrics (**Scoring**) that will be used for re-ranking the candidates using the lexicographic order of a vector of multiple similarity metrics, a or a linear combination of their values as describe in Section 3.3.3.

Matching

In this section we describe the matching algorithm applied between queries and initial candidates. We first formalize matching of subtrees based on their structure and then on their consistent re-labelling, providing support for unification of identifiers and constants (which is absent in the initial spectral matching performed by the Core Engine).

Notation: Define the formula tree T as either an OPT or SLT. The label on node n in T is denoted $\lambda(n)$. The number of nodes in T is denoted $|T|$. We also write $n \in T$ if n is a node in T and $(n_1, n_2) \in T$ if (n_1, n_2) is an edge in T.

Approximate matches of formulae might involve identifying corresponding parts of the trees that represent a query and a candidate match. We base such a correspondence on structural equivalence of those parts.[5]

Definition (*aligned SLTs/OPTs*): SLTs/OPTs T_1 and T_2 are *aligned* if there is an isomorphism f mapping nodes from T_1 onto nodes from T_2 such that for every edge $(n_a, n_b) \in T_1$, there is a corresponding edge $(f(n_a), f(n_b)) \in T_2$ that has the same label. (Note that *node* labels in aligned trees need not match.). We extend the definition to allow a node n in T_1 to map to a subtree t in T_2 if n has type *wildcard* and t is considered to be a single hypernode in T_2 with $\lambda(t) = t$. For N a subset of nodes in T_1, we define $f(N) = \{f(n) \mid n \in N\}$.

Approximate matches might also involve simple replacements of symbols in one SLT/OPT by alternative symbols (e.g., x for y or 3 for 2). Naturally, a wildcard symbol can be replaced by any symbol.

Definition (*unified nodes*): Node n_1 in SLT/OPT T_1 can be *unified* with node n_2 in SLT/OPT T_2, denoted $n_1 \dashrightarrow n_2$, if any of the following conditions holds:

- Both n_1 and n_2 have type *variable name* (V!),
- Both n_1 and n_2 have type *number* (N!),
- Both n_1 and n_2 have type *number* (F!),
- Both n_1 and n_2 have type *number* (T!),
- Both n_1 and n_2 have type *matrix* (M!),
- n_1 has type *wildcard* (*), and n_2 is a hypernode corresponding to any subtree in T_2, or
- $\lambda(n_1) = \lambda(n_2)$.

Next, when matching T_1 with T_2 and allowing substituted symbols, it is important that the substitutions are consistent when determining that T_1 and T_2 match approximately. We start by identifying candidate sets of nodes in T_1 that can be consistently relabeled.

Definition (*alignment partition*): Given T_1 and T_2, two aligned SLTs/OPTs with isomorphism f from T_1 to T_2, an *alignment partition* is a subset of nodes N in T_1 such that $(x \in N \wedge y \in N) \Rightarrow (\lambda(x) = \lambda(y) \wedge x \dashrightarrow f(x) \wedge y \dashrightarrow f(y) \wedge \lambda(f(x)) = \lambda(f(y)))$ (i.e., the nodes have identical labels and their unified images have identical labels or identical SLTs/OPTs). For node $n \in T_1$, we define $P(n)$ to be the alignment partition containing n if it exists and $\emptyset$ otherwise. (Note that $n \in P(n) \Leftrightarrow n \dashrightarrow f(n)$.) For alignment partition A, $\lambda(A)$ denotes the label that is common to all nodes in A and $\lambda(f(A))$ denotes the label that is common to all nodes in $f(A)$.

We can now choose a set of partitions that are consistent in their relabelling of nodes.

Definition (*matched set of nodes*): Given aligned SLTs/OPTs T_1 and T_2 with isomorphism f from T_1 to T_2 and the set of all corresponding alignment partitions, we define a *matched set of nodes* M as

$$\begin{aligned} M = \{n \in T_1 \mid\ & n \in P(n) \wedge \\ & \forall n' \in M([\lambda(n') = \lambda(n) \vee \lambda(f(n')) = \lambda(f(n))] \Rightarrow n' \in P(n))\} \end{aligned}$$

[5]Formally, a "part" of an SLT or OPT T will be a *pruned subtree*: any connected subset of labelled nodes from T together with the labelled edges connecting those nodes. Thus a pruned subtree of T is itself an SLT or OPT, but it need not extend to the leaves of T. Henceforth, we will use "subtree" to mean "pruned subtree."

In preparation to preferring matches of large *connected* parts of SLTs/OPTs, let $E(M) = \{(n_1, n_2) \mid n_1 \in M \wedge n_2 \in M \wedge (n_1, n_2) \in T_1\}$, the set of edges induced by M.

Note that there may be many possible matched sets of nodes for a given alignment, depending on which alignment partitions are chosen to be included. For aligned SLTs/OPTs T_1 and T_2 with isomorphism f from T_1 to T_2 and the set of all corresponding alignment partitions, we want to choose a matched set of nodes M that produces a high score, but evaluating all matched sets induced by an alignment is too expensive. Therefore we use a greedy algorithm that selects the partition including a set of matching nodes that maximizes the properties used for scoring.

SLT Wildcard Matching. We use a greedy approach based on three conditions that extend a wildcard to match as many nodes as possible as long as those nodes are not matchable by other nodes in the query. Assume that in some match for wildcard w, r is the root of $f(w)$. The first condition ensures that any axes emanating from r and not accounted for in the query are covered. The second condition extends the wildcard match from r along the *next* axis up to (but not including) the first node that can be matched exactly by the rest of query, or, failing that, up to (but not including) the first node that can be unified to a query node, or, failing both, to not include any node on the *next* axis. The third condition ensures that the wildcard match does not include a node emanating from r along the *next* axis if w has an edge to some node on another axis. Examples of different types of wildcard expansions behaviors are shown in Table 3.3. Greedy wildcard matching is thus formalized as follows:

Definition (*SLT greedy wildcard matching):* Given SLTs T_q and T_c and aligned SLTs T_1 in T_q and T_2 in T_c with isomorphism f, let node w be a node in T_q having type *wildcard*, let e_w be the set of labels on outedges from w, let $r \in T_2$ be the root of the SLT $f(w)$, let v_w^* be the sequence of nodes in T_c on the path starting at r having all edges labelled *next*, let f_w^* be the prefix of v_w^* within $f(w)$, and let $v_* \in f(w)$ be the last node on the path f_w^*. Given a node v, let $n(v)$ denote the node that is adjacent to v along the *next* axis (if such a node exists). *f obeys greedy wildcard matching* if all of the following conditions hold:

- edge $(r, v_L) \in T_c$ has label L and $L \notin e_w \Rightarrow$
  $f(w)$ includes the SLT rooted at v_L together with all its descendants in T_c
- $e_w = \{\mathit{next}\} \Rightarrow$
  $$(\lambda(n(w)) = \lambda(n(v_*)) \Rightarrow \nexists v \in f_w^*(\lambda(n(w)) = \lambda(v)))$$
  $$\wedge\ ((\nexists v \in v_w^*(\lambda(n(w)) = \lambda(v))\ \wedge\ n(w) \dashrightarrow n(v_*)) \Rightarrow \nexists v \in f_w^*(n(w) \dashrightarrow v))$$
  $$\wedge\ (\nexists v \in v_w^*(n(w) \dashrightarrow v) \Rightarrow (\nexists n(r) \vee n(r) \notin f(w)))$$
- $e_w \neq \emptyset\ \wedge\ e_w \neq \{\mathit{next}\} \Rightarrow$
  $\nexists n(r) \vee n(r) \notin f(w)$

OPT Wildcard Matching. We use a simpler definition of wildcard matching for OPTs as they can only appear as leaves of the tree. In this sense, the rules related to SLT wildcard contrainst based on their children nodes do not apply to OPT wildcards. Also, our current implementation of wildcards for OPT does not consider any kind of expansion across siblings of the matching candidate node as these would mean matching a single wildcard node to a set of subtrees from the candidate. An OPT wildcard will always expand to the children of the aligned candidate node.

Definition (*OPT greedy wildcard matching):* Given OPTs T_q and T_c and aligned SLTs T_1 in T_q and T_2 in T_c with isomorphism f, let node w be a node in T_q having type *wildcard*, let $r \in T_2$ be the root of the OPT $f(w)$, $f(w)$ includes the OPT rooted at r together with all its descendants in T_c.

Table 3.3: Tangent-S SLT greedy wildcard expansion. Wildcards are shown with red asterisks (e.g. $*$ or $*1*$). Exact matches are shown in green, wildcard matches in red and wildcard expansion anchors in blue. Symbols in black are unmatched.

Behavior	**Query**	**Match**
Unrestricted	$x + *$	$x+1$ $x+y+z+sin(x)$ $y+x+z=\frac{\pi}{4}$
	e^*	$f(x)=e^{x+1}+2$
Restricted by children	$*^2+1$	x^2+y^2+1 x^2+y+1 $x^2+(y+z)^2+1$
Restricted by binding	$*1*^2+*1*+1$	x^2+x+1 $(x+1)^2+(x+1)+1$ x^2+y+1
Horizontal expansion (right)	$x + *+1$	$x+y+1$ $x+y+z+1$ $x+y-z+1$ $x+\frac{1}{2+y}-3z+1$
Horizontal expansion (left)	$*+1$	$x+y+z+1$ $\alpha=f(x+y+1,x^2)$ $f(x,y)=\frac{1}{x+y+1}$

Finally, to compare a query SLT/OPT T_q against a candidate SLT/OPT T_c, we choose a pair of aligned subtrees that maximizes the score for the candidate with respect to the query. Thus for each node n in T_q, we consider the score of the largest subtree rooted at n that can be aligned with some subtree in T_c. We start by defining the largest alignable subtrees and use it as the basis for assigning a score.

Definition (*largest alignable subtrees*): Given SLTs/OPTs T_q and T_c, node $r_1 \in T_q$, node $r_2 \in T_c$, and aligned SLTs/OPTs T_1 rooted at r_1 and T_2 rooted at r_2 with isomorphism f, obeying greedy SLT/OPT wildcard matching, T_1 and T_2 are the *largest alignable subtrees rooted at* r_1 *and* r_2 if $r_1 \dashrightarrow r_2$ and there is no other subtree T_1' of T_q rooted at r_1 with an aligned SLT T_2' rooted at r_2 with isomorphism f' such that $|T_1'| > |T_1| \wedge (n \in T_1 \Rightarrow n \in T_1' \wedge f'(n) = f(n))$.

Definition (*largest common subtree*): Given SLTs/OPTs T_q and T_c, consider pairs of nodes $n_1 \in T_q$ and $n_2 \in T_c$ such that n_1 can be unified with n_2. Let T_{i_1} and T_{i_2} be the largest alignable subtrees rooted at n_1 and n_2. The *largest common subtree* of T_c with respect to T_q is $\max_i s(T_q, T_c; T_{i_1}, T_{i_2}, M_i)$ over all such pairs, where M_i uses our greedy matching algorithm to chose matched sets of nodes.

Note that in the case of OPTs, the order of arguments for commutative operators is ignored to capture matches between equivalent expressions such as $x+y=0$ and $0=y+x$. However, testing all possible permutations of children at matching time has a factorial time complexity. We use a greedy pair-wise matching algorithm that considers all pair-wise alignments between children of matching commutative operators, and greedily chooses 1-to-1 matches between children maximizing the predicted number of matches after unification, breaking ties by preferring alignments with more exact matches. While suboptimal, this greedy approach can still be computed in polynomial time because only one permutation is evaluated in every case, and it still allows $x + y = 0$ to match $0 = y + x$ perfectly.

Scoring

In this section, we define multiple similarity metrics based on matched subtrees. These can be applied to score a candidate SLT/OPT against a query that may include wildcards. Because the SLT/OPT for an arbitrary query formula will not necessarily align with the SLT/OPT for an arbitrary candidate match formula, we need to consider subtrees of the SLTs/OPTs that can be aligned. In so doing, we need to allow (but penalize) situations in which superfluous or mismatched symbols might appear in the query or in the candidate match. We wish to balance the amount of structural match with the number of symbols that are identically preserved, but also take into account the size of the "excess" unmatched structure and the extent to which wildcards engulf candidate nodes.

We start with query tree T_q and candidate tree T_c. Let $Pairs(x, w)$ be a function that returns the set of symbol pairs defined by a set of nodes x given the window parameter w. In this case x can be a subset of nodes from either T_q or T_c. The initial ranking score provided by the core engine can be defined as follows:

Definition (*Approximated Dice Coefficient score (ADC)*): Represents the metric used by the Core Engine for selection of candidates. It is the approximated Dice coefficient of recall and precision of matched symbol pair tuples

$$R_1 = \frac{|Pairs(T_q, w) \cap Pairs(T_c, w)|}{|Pairs(T_q, w)|}, \qquad P_1 = \frac{|Pairs(T_q, w) \cap Pairs(T_c, w)|}{Pairs(T_c, w)}$$

$$\mathbf{ADC} = \frac{2R_1P_1}{R_1 + P_1}$$

The metric is considered an approximation because wildcards are not expanded beyond a single symbol, so additional symbols that could be covered by wildcards are considered unmatched by this metric. When the candidate expression and/or query expressions have a depth less than two, this metric considers additional tuples for symbols at the end of writing lines (EOL). Also, note that the core engine does not perform unification.

After aligning query and candidate trees using our matching algorithm, we define M as the set of nodes from T_q that have been matched by nodes from a candidate tree T_c. Let T_m be a pruned subtree of T_c defined by the nodes matched by M. We can split M into three mutually exclusive subsets M_q^E, M_q^U, and M_q^W representing exact, unified and wildcard matched nodes from the query respectively. Their counter parts are the sets M_c^E, M_c^U, and M_c^W representing exact, unified and wildcard matched nodes from the candidate respectively.

We then consider a constrained version of the ADC score after matching as follows:

Definition (*Constrained Dice Coefficient score (CDC)*): This is a modified version of the Approximated Dice Coefficient (ADC). Unlike the original metric, this constrained version is computed after finding the largest common subtree using M. As a result, many inconsistent tuple

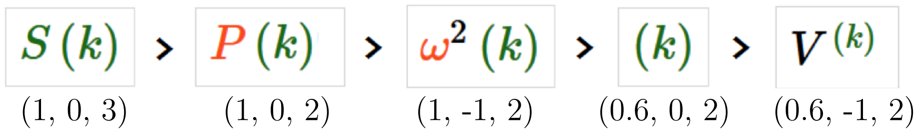


Figure 3.9: MSS Scoring for Query $S(k)$. Ranking triples contain MSS (1), and the number of query symbols that are unmatched (2) and exactly matched (3). Parentheses count as one symbol.

matches will be removed resulting in lower scores for some expressions that ADC would rank higher. However, matching considers wildcard expansion resulting in higher scores for some candidates that ADC would rank lower. The unification algorithm is disabled for this metric. In addition, this metric does not require the usage of additional tuples for symbols at the end of writing lines, instead it adds a dummy pair to the count to deal with small expressions. The CDC score is defined as:

$$R_2 = \frac{|Pairs(M_q^E \cup M_q^W, w)| + 1}{|Pairs(T_q, w)| + 1}, \quad P_2 = \frac{|Pairs(M_c^E \cup M_c^W, w)| + 1}{|Pairs(T_c, w)| + 1}$$

$$\mathbf{CDC} = \frac{2R_2P_2}{R_2 + P_2}$$

We then consider an alternative version using variable unification as follows:

Definition (*Dice Coefficient with Unification (UDC)*): Similar to CDC, this is also based on the Dice coefficient of matched symbol pairs after matching. However, this metric considers unification during matching and scores unified matches lower than exact matches. The UDC is computed as follows:

$$R_3 = \frac{Pairs(M_q^E \cup M_q^W, w) + Pairs(M_q^E \cup M_q^U \cup M_q^W, w) + 2}{2(Pairs(T_q, w) + 1)}$$

$$P_3 = \frac{Pairs(M_c^E \cup M_c^W, w) + Pairs(M_c^E \cup M_c^U \cup M_c^W, w) + 2}{2(Pairs(T_c, w) + 1)}$$

$$\mathbf{UDC} = \frac{2R_3P_3}{R_3 + P_3}$$

Finally, we consider that an ideal scoring function should have the following properties, as illustrated in Figure 3.9: alignments with more matched symbols, and especially identical symbols, in close proximity to each other score higher than those with fewer matched symbols or more disconnected matches; if two candidates score equally with respect to matched symbols and their proximity, the one with fewer superfluous symbols scores higher; and everything else being equal. Alignments with more matched symbols that are identical scores higher. For this reason, we define the Maximum Subtree Similarity score as follows:

Definition (*Maximum Subtree Similarity score (MSS)*): Let the Maximum Subtree Similarity score MSS be the harmonic mean of the fraction of nodes from T_q preserved by M and the fraction of edges preserved by $E(M)$:

$$\mathbf{MSS} = \begin{cases} \frac{2}{\frac{|T_q|}{|M|} + \frac{|T_q|-1}{max(|E(M)|, 0.5)}} & if |M| > 0 \\ 0 & otherwise \end{cases}$$

this is a measure of the size of the consistent structural part of the match.

Intuitively, *MSS* is used to rank more structurally similar query matches on a candidate higher, on the assumption that they are more likely be perceived as relevant by users. Were this true, ordering by *MSS* would be consistent with the Probability Ranking Principle [123], which states that hits are ideally sorted in decreasing order of probable relevance to the query, $P(c|q)$. In the future, one might examine the correlation of *MSS* with $\hat{P}(c|q)$ estimated from human relevance ratings.

Multiple similarity scores are considered in order to break ties and also to account for other potential factors of similarity between formulas. In the re-ranking produced by this layer of the system, the scores assigned to any two candidate matches are compared lexicographically to determine which candidate ranks higher. Below we define different additional scores that we have considered during our experiments.

Definition (*Unmatched score*): This similarity score represents the total number of elements that are unmatched in T_c after unification and wildcard expansion. It can be represented as the negation of the number of unmatched nodes in T_c, i.e., $\mathit{Unmatched} = |M| - |T_c|$.

Definition (*Exact score*): Represents the number of query elements that were matched exactly (not unified), $\mathit{Exact} = |M_q^E| + |M_q^W|$. Query wildcard matches are considered part of the exact matches in this case, but even if they are matched to very large subtrees, they still count for exactly one query element.

Definition (*Wildcard score*): In some cases, we might prefer matches where less elements in the candidate need to be matched by wildcards. The wildcard score is given by the negation of the number of elements matched by wildcards, $\mathit{Wilcard} = -M_c^W$

Definition (*Wildcard Standard Deviation score (σ^W)*): This score prefers matches where wildcards match subexpression of uniform sizes. For example, for a query with 2 wildcards, this score prefers a candidate where each wildcard matched 5 elements to another candidate where the first wildcard matched 2 elements and the second 8. The score σ^W is given by the negation of the standard deviation of the size of the matches for each matched query wildcard node.

Definition (*Left-position scores*): For SLTs, let *LeftPos* be a score that depends on the left to right location of the root of T_m in T_c. *LeftPos* will give preference to subtrees located closer to the root of T_c. More than one score can be used to further distinguish between elements matched in secondary baselines, where the first score refers to the position of the match in the main baseline, then the second score refers to the position on the secondary baseline and so on.

For example, if we use two positional scores, then for query x and candidate $2 + x = 0$, the scores become $(3, 0)$ indicating that the match was located at the third element of the main baseline and no secondary baseline was used. For another candidate $2 + y^{3+3} = 0$, the scores become $(3, 3)$ indicating that the match is appears on a children of third element of the main baseline and then, that the match is located at the third element within that secondary baseline. The negation of these scored is used for lexicographic ordering.

Similarity Metrics Vectors. We define four particular combinations of similarity scores that we have tested for isolated formula retrieval.

Definition (SM1 - *Approximated Dice Coefficient*): Represents the metric used by the Core Engine for selection of candidates with tie breakers:

$$\mathbf{SM1} = (ADC)$$

Definition (SM2 - *Constrained Dice Coefficient*): Uses the Constraint Dice Coefficient (CDC) as the main similarity score, and three tie breakers:

$$\mathbf{SM2} = (CDC, \mathit{Wildcard}, \sigma^W, \mathit{LeftPos})$$

Definition (SM3 - *Dice Coefficient with Unification*): Uses the Dice Coefficient after Unification as the main similarity score, and three tie breakers:

$$\mathbf{SM3} = (UDC, \mathit{Wildcard}, \sigma^W, \mathit{LeftPos})$$

Definition (SM4 - *Maximum Subtree Similarity*): Uses the Maximum Subtree Similarity as the main score and two tie-breakers. An example of the sorting produced by this metric is given in Figure 3.9. The metric is defined as follows:

$$\mathbf{SM4} = (MSS, \mathit{Unmatched}, \mathit{Exact})$$

Definition (SM5 - *Normalized Maximum Subtree Similarity*): Uses the Maximum Subtree Similarity as the main score and two normlized tie-breakers. The final sorting is very similar to **SM4** in most cases. The metric is defined as follows:

$$\mathbf{SM4} = (MSS, \mathit{UnifiedPrecision}, \mathit{ExactRecall})$$

where *UnifiedPrecision* is the precision after unification, and *ExactRecall* is recall for exact and wildcard matches without unification.

3.3.3 Layer 3: Linear Regression Re-ranking

If relevance judgments data is available, it is possible to use it to train a regressor that predicts relevance scores using the similarity scores computed in the second layer of the system (Section 3.3.2). Using relevance judgments data from the NTCIR-12 Wikipedia Formula Retrieval task, we train a least squares linear regressor to predict a final rank score using a combination of three scores: MSS, precision after unification, and recall of exact matches. While more complex functions could have been used in this step, we choose a simple method to avoid over-fitting the limited training data available, and to clearly observe which re-ranking scores best predict relevance.

3.3.4 Combining Formula Representations

We combine results from SLTs and OPTs in a simple way as illustrated in figure 3.3. First, we perform symbol pair-based retrieval within a separate index for each representation. The top-k candidates obtained from each index are merged into a single list. Then, for each candidate we apply the detailed *matching* and *scoring* processes using both SLTs and OPTs representations, and we concatenate the similarity scores into a single vector. Finally, a linear regressor assigns a relevance score for the formula. In our experiments, we see that this simple combination obtains better rankings than using scores from just SLTs or OPTs.

3.3.5 Combining with Keywords

The text retrieval component of Tangent-3 is unchanged from Tangent-2 [88]. We have used Solr which provides an inverted index and TF-IDF scoring algorithm. The text similarity score used in our system is provided by Solr's Disjunction query, which applies TF-IDF to multiple fields of a document. For a single field, Solr scores a document d for query $q = \{t1, ..., tn\}$ by:

$$s_t(q,d) = c(q,d)idn(q)\sum_{t\in q}\sqrt{freq(t,d)\cdot idf(t)^2}\cdot norm(t,d)$$

where c is the ratio of query terms (t_i) matched in the document, $idn(q) = \sqrt{\sum_{t\in q} idf(t)^2}^{-1}$ normalizes the squared *idf* values, $freq(t,d)$ the number of times t appears in d, and $norm(t,d)$ is the normalized number of tokens in the field (in our case, body text or title).

To emphasize title text, the score for the title field is doubled, and then the maximum of the title and body text scores is returned as the final text score.

After searching for keywords on the text index and finding documents with matching formulas, these outputs are combined into a single final ranking of documents. For this task, we compute the final score s_f of a document d for query q using a simple linear combination as follows:

$$s_f = \alpha\left(\sum_{e\in q} bestmatch\,(e,d)\,w_e\right) + (1-\alpha)\,s_t\,(q,d)$$

where α is the weight given to the math component of the query, $bestmatch(e,d)$ is a function that returns the score for the best matching formula in d for math expressions e, w_e is the relative weight given to math expression e. $s_t(q,d)$ represents the text search score for document d. If there are no keywords on the query q, then $s_t(q,d)$ is equal to 1. Note that $\sum_{e\in q} w_e = 1$ and $0 \leq \alpha \leq 1$.

The function $bestmatch(e,d)$ returns a score vector with m dimensions as defined in Section 3.3.2. The result from s_t is treated as a vector with the actual text score value on the first dimensions and 0 for each of the remaining m - 1 dimensions. The final score of a document is a vector of scores. Documents are sorted by lexicographical order of these score vectors.

3.4 Experiments

The proposed formula retrieval approach has been tested using multiple configurations on different standard datasets. In this section, we describe the three major sets of experiments used for evaluation and improvements of our model. The first set of experiments corresponds to the NTCIR-11 Math-2 Retrieval Task [3] and focuses on the Wikipedia Subtask as a way to test the recall of our system, the NTCIR-11 arXiv dataset as a way to test the scalability of the model, and also includes a test with human evaluators. The second set of experiments corresponds to our submissions to the NTCIR-12 MathIR Task [136] and evaluates the performance of different configurations of the system for queries with and without text. Finally, the third set of experiments uses the same data collections from NTCIR-12 MathIR task, and focuses on the optional Wikipedia Formula Browsing Task to evaluate the performance for our system using both SLTs and OPTs.

3.4.1 Experiment Set 1: NTCIR-11 Math-2 Retrieval Task Results

In this section we present experiments designed to observe the effect of system parameters on index size, retrieval time, and search results, along with a human assessment of top-10 results. Only the

SLT representation was used for these experiments.

The main dataset used is the NTCIR-11 Wikipedia collection [3] with 30,000 articles totalling 2.5 GB and containing roughly 387,947 unique LaTeX expressions. In addition, we use the much larger NTCIR-11 arXiv collection to test the scalability of Tangent-3. The arXiv collection is 174 GB uncompressed, with 8,301,578 documents (arXiv article fragments) and roughly 60 million formulae including isolated symbols. We use the Wikipedia data set to study the retrieval behavior of the core engine, and the arXiv dataset to test the scalability of the system. For our computer-based experiments, we use the 100 formula queries from the Wikipedia collection for both data sets, and a subset of the Wikipedia queries for our human experiment, shown in Figure 3.11.

In this early set of experiments, we do not consider wildcard expansion, and we only evaluate our formula retrieval model using the first two layers: Core-Engine and formula Re-ranking after matching and scoring by **SM4**: *MSS*, *Unmatched*, and *Exact* scores. Formulas are sorted using the lexicographic ordering of these three scores.

Efficiency

Computational Resources and Parameters. We use a Ubuntu Linux 14.04 server with 24 Intel Xeon processors (2.93GHz) and 96GB of RAM. While some indexing operations were parallelized (as noted below), *all retrieval times are reported for single-threaded processing.* Parallelization of query execution, and parallelizing re-rank scoring over matching formulas could be used to further speed up processing, and additional opportunities for decreasing run-times are discussed below. All results reported for Tangent-3 in this Section were obtained using the top 100 formula from the core engine (i.e., $k = 100$).

Indexing. As seen in Table 3.4, index size increases roughly linearly from window sizes 1-4, with end-of-line tuples increasing storage by a constant amount. Adding just EOL tuples for small expressions (Sm-EOL; see Section 3.2) increases the index size modestly (by less than 500k for Wikipedia, and less than 10 MB for arXiv). The maximum Wikipedia index size is 503.1 MB on disk; in contrast, for the arXiv the maximum index size is 29 GB. For small window sizes storage is much smaller; for $w = 1$ with *No-EOL* and *Sm-EOL*, the index file is just over 64 MB for Wikipedia, and just under 5.4 GB for arXiv; these are much smaller than Tangent-2 (1.3 GB for Wikipedia, and roughly 36 GB for the arXiv dataset [88]). When these index files are loaded into memory, they consume 2 - 2.5 times their space on disk.

For the arXiv data, it took 43 hours to pre-process the documents (using 10 processes), and at most an additional 3.5 hours to generate the index (when $w = All$ and end-of-line tuples are included) using a single process. Wikipedia was much faster, requiring 260 seconds for preprocessing, and at most 95 seconds for index creation. As our document pre-processor is implemented in Python, we believe that a faster implementation (e.g., in C++) could reduce run times by a factor of 4-10 in both cases.

Retrieval Times. We ran the 100 NTCIR-11 Wikipedia formula queries over the large NTCIR-11 arXiv collection to test retrieval speed for the core engine (See Figure 3.10). Retrieval times are directly affected by the window size and EOL parameters. As expected, the increased number of index entries produced by larger window sizes lead to longer query execution times. For example, when EOL tuples are ignored, retrieval times in milliseconds (given as (μ, σ, median)) increase from (372.95, 1649.24, 90.40) for $w = 1$ to (1932.41, 9332.54, 281.64) for $w = All$. Including EOL pairs had a larger effect on performance. For $w = 1$, retrieval times increase to (4183.79, 15705.23, 575.54) when EOL pairs are included. However, using Sm-EOL increases retrieval times only a small amount over when no EOL pairs are used (435.55, 1626.48, 111.39).

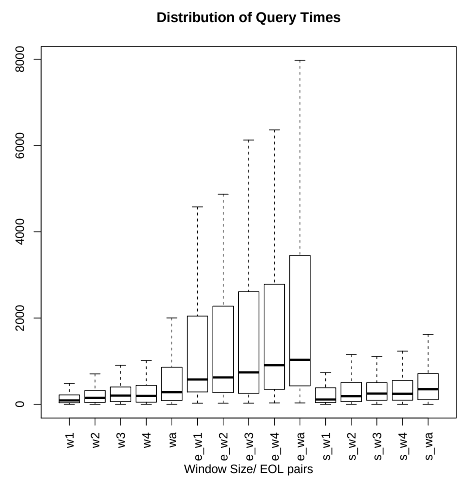


Figure 3.10: Core Engine Retrieval Times (in *ms*) for NTCIR-11 arXiv Corpus (60 million formulae). *w* denotes window size, *e* the addition of End-of-Line (EOL) tuples, and *s* adding Small-EOL tuples.

Table 3.4: Index Sizes for NTCIR-11 Collections.

	Index Sizes (*MB*)					
	Wikipedia			arXiv		
w	No-EOL	Sm-EOL	EOL	No-EOL	Sm-EOL	EOL
1	64.2	64.5	73.7	5,364	5,372	6,179
2	95.6	96.0	105.2	7,568	7,577	8,383
3	128.3	128.7	137.9	9,662	9,671	10,477
4	161.3	161.7	170.9	11,587	11,596	12,402
All	493.5	493.9	503.1	28,225	28,234	29,040

When running the same queries on the much smaller NTCIR-11 Wikipedia collection, using parallel search of nine sub-indices Tangent-2 requires 8 *minutes* to complete the search for the 100 test queries. In stark contrast, Tangent-3 requires only 8.48 *seconds* without re-ranking, and 106.14 seconds with re-ranking for the slowest condition ($w = All$, EOL), and 0.57 seconds without re-ranking and 78.03 seconds with re-ranking for the fastest condition ($w = 1$, no EOL).

Re-ranking times are consistent across w and *EOL* settings because the number of formulae re-ranked is fixed at $k = 100$. For the Wikipedia task, re-rank times are (median, μ, σ) = $(72, 775, 3562)$ ms. The mean is skewed by a small number of outliers. In one extreme case, re-ranking takes 46 seconds, with retrieval from the core ($w = All$ with EOL) taking only 1.7 seconds. This particular expression is very large, with 16 wildcard symbols (*Query 52*), producing large hits with many possible unifications. We do not expect to see many queries of this type in common use. Re-rank times can be improved substantially if the re-ranker were recoded from Python to C++.

Effectiveness

The NTCIR-11 Wikipedia benchmark [105] includes 100 queries for measuring specific-item retrieval performance. Queries are associated with a single target formula in a specific document, ignoring identical formulae appearing within the same or different documents. These 100 queries are split into 65 *Constant* queries containing no wildcards and 35 *Variable* queries containing wildcards.

Search results are returned as a ranked list of (*documentId*, *formulaId*) pairs. *Document-*

Table 3.5: NTCIR-11 Wikipedia Formula Retrieval Benchmark Results. (100 Queries: 65 Constant, 35 Variable (w. wildcards)). Metrics: % recall for targets (top-10,000) and Mean Reciprocal Rank (MRR, in %).

	RECALL						MRR					
	DOCUMENT-CENTRIC			FORMULA-CENTRIC			DOCUMENT-CENTRIC			FORMULA-CENTRIC		
Participant	Total	Const	Var	Total	Const	Var	Total	Const	Var	Total	Const	Var
TUW Vienna †	**97**	100	91	**93**	98	83	**80**	80	79	**82**	86	75
NII Japan †	**97**	98	94	**94**	97	89	**74**	79	74	**72**	87	63
Tangent-2 (RIT)	**88**	91	83	**78**	78	77	**70**	68	75	**67**	65	72
Tangent-3 Core ∘												
w=1, No-EOL	**95**	97	91	**95**	97	91	**79**	80	77	**76**	76	76
Sm-EOL	**97**	100	91	**97**	100	91	**80**	82	77	**77**	78	76
EOL	**98**	100	94	**98**	100	94	**81**	82	79	**77**	78	76
w=All, No-EOL	**95**	97	91	**95**	97	91	**79**	80	78	**76**	76	76
Sm-EOL	**97**	100	91	**97**	100	91	**80**	82	78	**78**	78	76
EOL	**97**	100	91	**96**	100	89	**81**	82	78	**77**	78	75
Tangent-3 Re-rank ∘												
w=1, No-EOL	**95**	97	91	**95**	97	91	**80**	80	82	**77**	76	80
Sm-EOL	**97**	100	91	**97**	100	91	**82**	82	82	**79**	78	80
EOL	**98**	100	94	**98**	100	94	**82**	82	82	**79**	78	80
w=All, No-EOL	**95**	97	91	**95**	97	91	**80**	80	80	**77**	76	77
Sm-EOL	**97**	100	91	**97**	100	91	**81**	82	80	**78**	78	77
EOL	**97**	100	91	**96**	100	89	**82**	82	82	**78**	78	78

†: uses Operator Tree (OT) formula representation (Content MathML)
∘: uses Symbol Layout Tree (SLT) formula representation (Presentation MathML)

centric results are computed using the list of document identifiers in their order of appearance after removing duplicates. *Formula-centric* results are computed using the complete ranked list of matches. Note that if the query formula is found at a different location within the target document, this is considered a miss.

Systems are evaluated using two metrics. First, by the *recall* (i.e., percentage of targets located at rank 10,000 or less) and second by the *mean reciprocal rank* (MRR) over all queries (assigning zero when not found). Note that a reciprocal rank of 50% indicates that on average, the target formula appears at rank two.

Summary of Results. At the top of Table 3.5, NTCIR-11 results from the four best systems are shown as fractions rounded to the nearest percentage [105]. A summary of participating systems is available [3]. The first three systems are *tree-based* approaches applied to Operator Trees encoded in Content MathML. The fourth system is a *spectral* approach, using symbol pair-based retrieval over Symbol Layout Trees encoded in Presentation MathML. TUW Vienna indexes individual symbols and linearized subexpressions. NII represents sub-expressions in Operator Trees by hash code sets (see Section 2.1). TUB Germany makes use of XQuery expressions applied to Operator Trees.

Tangent-3 results are shown at the bottom of Table 3.5, split into core and reranked results. We include combinations of window sizes $w = \{1, 2, 3, 4, \mathit{All}\}$ (where *All* is all tuples) and End-of-Line symbol indexing settings (No-EOL, Small-EOL, EOL).

Using re-ranking, $w = 1$ and all EOL tuples, Tangent-3 obtains the highest document and formula recall (both 98% vs. 97% and 94%), the highest Variable (wildcard) formula MRR (80%) and the highest document MRR (82%) to date. In all Tangent-3 conditions, the mean rank of a target formula is just above the middle of ranks one and two (i.e., higher than an MRR of 75%). This is interesting, as previously the strongest results make use of Operator Trees rather than SLTs for retrieval. This supports the idea that choice of formula representation may be less important than the choice of primitives for retrieval. The higher formula recall obtained by Tangent-3 may be due to matching symbol pairs rather than complete subtrees (see Section 2.1).

Relative to the previous version of the Tangent model, Tangent-2 [88], recall is increased 10% for documents, and 20% for formulae. MRR values are also 10% higher in Tangent-3 (all conditions) than Tangent-2.

Using *Sm-EOL* provides state-of-the-art performance comparable to *EOL*, but uses a smaller index size with faster retrieval times. With *Sm-EOL* we obtain document and formula recall of 97% (a reduction of only 1%) with unchanged MRR values. For the same parameter settings, formula MRR is 80% (the best reported value is 82%).

Window Size. Window size had little effect on performance, and so for space we show only results for $w = 1$ and $w = All$ in Table 3.5. Interestingly, recall in the top-10,000 actually decreases slightly when $w = All$ is used, missing two additional formula with wildcards (Var) and one document relative to $w = 1$. This may be because of noise introduced by the larger number of tuples, which may match anywhere in a candidate formula. Window size had no effect on MRR for queries without wildcards ($Const$), but we again see a small decrease for queries with wildcards for $w = All$.

End-of-Line Symbols. For $w = 1$, adding all EOL tuples increases the number of formulae retrieved by three (e.g., the query ‘s’ produces no symbol pairs and thus requires an EOL tuple if it is to be matched). *Sm-EOL* gave fast queries and also retrieved two of the three formulae missed when EOL tuples are omitted. For $w = All$ using all EOL tuples decreases formula recall slightly relative to *Sm-EOL* (missing one query with wildcards), perhaps because of the numerous matches obtained using $w = All, EOL$. Adding the *Sm-EOL* tuples increases document and formula MRR slightly for non-variable queries.

Re-ranking. The re-ranker does not affect recall for Tangent-3, as the re-ranker only reorders hits returned by the core. Exact matches are represented well by the Dice coefficient over tuples, particularly for concrete queries without wildcards. However, the re-ranker does improve wildcard (variable) formula MRR by 4% for queries with wildcards for $w = 1$, *Sm-EOL*. (80% with re-ranking, vs. 76% without). This is 5% higher than the best previous MRR results for formula MRR (75%).

On average, only 4.16 of the top-10 results (with standard deviation of 2.61) remain in the top-10 after re-ranking by MSS. MSS will not re-order exact or very nearly exact matches, hence constant queries have the same MRR values as the core engine, but the reranker is able to improve MRR values for variable queries where the core implements approximate wildcard handling.

Without re-ranking, the core engine still produces comparable results using $k = 100$. For the core engine, increasing k produces perfect recall (100%) without substantially increasing query time. The reranker, however, cannot process a large number of results in a reasonable amount of time. The core engine also produces higher document-centric MRR values than the best published results, although limited handling of wildcards causes variable queries to underperform.

Human Evaluation of Retrieval Results

Having seen that Tangent works well for retrieving specific formula, we now consider how well MSS-based rankings relate to human perceptions of formula similarity, using the top-10 results from 10 of the NTCIR-11 Benchmark queries (see Figure 3.11).

Data. 10 queries were selected using random sampling from the Wikipedia queries. Five contained wildcards, and the other five did not. (see Figure 3.11). Some queries were replaced using a new randomly selected query to ensure a diverse set of expression sizes and structures.

To limit the number of hits for participants to evaluate, we chose to consider only $w = 1, 2$, and *All*. We used *No-EOL*, as none of the queries were a single symbol (the smallest query, Q8 is ‘$\alpha(x)$’), and we wished to observe the effect of window size using actual symbol pairs, without EOL

Familiarization Queries	
$z * x \leq y$	(10)
T_*	(33)

Concrete Evaluation Queries	
$\alpha(x)$	(8)
$bx - x^2$	(15)
$E\left[\hat{\sigma}^2\right] = \frac{n-1}{n}\sigma^2$	(53)
$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy = pdx + vdy$	(70)
$(\sqrt{p_1}, \cdots, \sqrt{p_n})$	(95)

Evaluation Queries with Wildcards	
$Z = \sum_j g_j \cdot \mathrm{e}^*$	(19)
$(x+y)^n = \sum_*^n \binom{n}{k} x^{n-k} y^k = \sum_*^n \binom{n}{k} x^* y^{n-k}.$	(39)
$\frac{D_g u_*}{Dt} - f_0 v_a - \beta y v_g = 0$	(44)
$p = \frac{-x \pm \sqrt{x^* - 4(*)(\frac{*}{*} - y)}}{2(\frac{-gx^2}{*})}$	(60)
$\pi_i = 2^* \binom{N}{i}$	(94)

Figure 3.11: Queries for Human Evaluation, with NTCIR-11 Wikipedia Query Numbers Shown

Table 3.6: Likert Rating $\mu(\sigma)$ for Top-10 NTCIR-11 Wikipedia Hits (21 participants, 10 queries).

	Rank/Position in Top-10 Hits									
w	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	4.54 (0.78)	3.79 (1.16)	3.48 (1.31)	3.23 (1.30)	2.83 (1.25)	2.94 (1.22)	2.65 (1.19)	2.78 (1.21)	2.78 (1.21)	2.85 (1.25)
2	4.54 (0.78)	3.71 (1.22)	3.48 (1.30)	3.16 (1.28)	2.90 (1.26)	2.93 (1.20)	2.85 (1.25)	2.57 (1.18)	2.74 (1.22)	2.80 (1.13)
All	4.54 (0.78)	3.78 (1.19)	3.59 (1.23)	3.27 (1.16)	2.98 (1.25)	2.92 (1.23)	2.80 (1.17)	2.98 (1.24)	2.92 (1.21)	2.87 (1.17)
best	4.54 (0.79)	4.03 (1.07)	3.75 (1.12)	3.49 (1.13)	3.31 (1.20)	3.15 (1.16)	3.02 (1.17)	2.94 (1.17)	2.85 (1.19)	2.77 (1.19)

tuples. Using the Wikipedia collection, for the three versions of the core compared ($w = \{1, 2, All\}$, *No-EOL*), we applied re-ranking to the top-100 hits, and then collected the top-10 hits returned by each query for rating.

Evaluation Protocol. Participants completed the study alone in a private, quiet room with a desktop computer running the evaluation interface in a web browser. The web pages provided an overview, followed by a demographic questionnaire, instructions on evaluating hits, and then familiarization trials (10 hits; 5 for each of two queries). After familiarization, participants evaluated hits for the 10 queries, shown in Figure 3.11 and finally completed an exit questionnaire. Participants were paid \$10 each at the end of their sessions.

Participants rated the similarity of queries to results using a five-point Likert scale (1 = Very Dissimilar, 2 = Dissimilar, 3 = Neutral, 4 = Similar, 5 = Very Similar). It has been shown that presenting search results in an ordered list influences relevance assessments [35]. Instead we presented queries along with each hit in isolation, with query presentation order randomized, and the presentation order for hits was also randomized.

Demographics and Exit Questionnaire. 21 participants (5 female, 16 male) were recruited from the Computing and Science colleges at RIT. Their age distribution was: 18-24 (8), 25-34 (9), 35-44 (1), 45-54 (1), 55-64 (1) and 65-74 (1). Their highest levels of education completed were: Bachelor's (9), Master's (9), PhD (2), and Professional (1). Their reported areas of specialization were: Computer Science (13), Electrical Engineering (2), Psychology (1), Sociology (1), Mechanical Engineering (1), Computer Engineering (1), Math (1) and Professional Studies (1).

In the post-questionnaire, participants rated the evaluation task as Very Difficult (3), Somewhat Difficult (10), Neutral (6), Somewhat Easy (2) or Very Easy (0). They reported different approaches to assessing similarity. Many considered whether operations and operands were of the same type or if two expressions would evaluate to the same result. Others reported considering similarity primarily based on similar symbols, and shared structure between expressions.

Similarity Rating Results. As seen in Table 3.6, the mean similarity rating distributions

Table 3.7: Likert Rating nDCG@10 for Wikipedia Queries (vs. Pooled Top-10 Ratings for $w = 1, 2, \mathit{All}$).

	CONSTANT					
w	**Q08**	**Q15**	**Q53**	**Q70**	**Q95**	
1	**0.95**	0.86	**0.90**	**0.89**	0.79	
2	0.94	0.87	0.88	0.87	0.88	
All	**0.95**	**0.90**	0.88	0.85	**0.90**	
Hybrid	**0.95**	**0.90**	**0.90**	**0.89**	**0.90**	
	VARIABLE					CONST. + VAR.
w	**Q19**	**Q39**	**Q44**	**Q60**	**Q94**	$\mu(\sigma)$
1	**0.93**	**0.95**	0.84	0.62	**0.93**	0.87 (0.10)
2	0.91	**0.95**	**0.92**	0.67	0.88	0.88 (0.08)
All	0.89	0.90	0.91	**0.93**	0.92	**0.90 (0.03)**
Hybrid	**0.93**	**0.95**	**0.92**	**0.93**	**0.93**	**0.92 (0.02)**

are very similar in all conditions as measured by the normalized discounted cumulative gain at top 10 (nDCG @ 10). Average ratings increase consistently from the 5th to 1st hits and are close to the best of the measured query results. The top 4 hits have an average rating higher than '3,' suggesting that participants felt these formulae had some similarity with the query expression. As matches were not highlighted, sometimes even exact matches were rated as '4' rather than '5' (e.g. for large matching formulae).

The top-5 hits are largely identical across conditions (a minimum of 4/5 hits in agreement), with the exception of query 60: $p = \frac{-x \pm \sqrt{x^{*} - 4(*)(\frac{*}{*} - y)}}{2(\frac{-gx^2}{*})}$, where $w = \mathit{All}$ retrieves all the top 5 rated hits but for $w = \{1, 2\}$ only 1 of the best-ranked pooled matches is among the top-5 hits. Here $w = \mathit{All}$ generates more tuples for symbols that are leftmost in the formula, and compensates for the fact that wildcard symbol pairs are not indexed through relationships between more distant symbols.

To further investigate the effect of window size on ranking behavior, we pooled the top-10 hits for each query, and then sorted by each hits' average Likert rating to produce an ideal ranking (see Table 3.7). Relative to this ideal ranking, average nDCG@10 values increase and standard deviations decrease with window size ($w = 1$: 0.87 (0.1); $w = 2$: 0.88 (0.08); $w = \mathit{All}$: 0.90 (0.03)). Using an oracle to select the best window size for each query improves nDCG and decreases variance (*oracle*: 0.92 (0.02)). Interestingly, the highest nDCG values are obtained for six of the queries using $w = 1$. This suggests that ideal window size may be query-dependent (e.g., we may need larger window sizes to handle many wildcards, as for query 60).

3.4.2 Experiment Set 2: NTCIR-12 MathIR Task

In this section, we describe our submissions to the NTCIR-12 MathIR Task [136] using Tangent 3.1 (see Table 3.1) . The competition provided an opportunity to further test configurations and some refinements to the initial model of Tangent-3 [138]. The most important refinements done to the initial formula retrieval model was the inclusion of the greedy wildcard expansion matching described in section 3.3.2. Only the SLT representation was used for these experiments. Four alternative similarity metrics used for computing the final rankings of formulas (see Section 3.3.2). Also, the main task included queries containing multiple formulas and keywords. We explore four different ways to combine partial search results for such queries as described in Section 3.3.5.

Different configurations of the Tangent-3 system [27] were submitted to three out of four tasks defined by the NTCIR-12 MathIR competition [136]: arXiv Main Task, optional Wikipedia Task, and optional Wikipedia Formula Browsing Task.

The NTCIR-12 MathIR main arXiv Task used the same collection from NTCIR-11 Math-2 Retrieval Task that contains more than 60 million formulae as described in Section 3.4.1. The task defines a set of 29 queries containing at least one math formula expression, and most of them contain one or more keywords.

For the optional Wikipedia tasks, a corpus of 319,689 articles from English Wikipedia with more than half a million formulae were used. The NTCIR-12 MathIR optional Wikipedia Task defines 30 queries containing at least one formula, and most of them contain at least one keyword. Also, many of these queries are defined in natural language as actual questions related to different topics in math. Finally, the NTCIR-12 MathIR optional Wikipedia Formula Browsing task defines 40 queries for isolated formula retrieval, where 20 are concrete (with no wildcards), and the other 20 are derived from the first 20 by replacing specific portions of the formulae by wildcards.

In order to investigate how to assign weights to keywords and math expressions, we tested two different ways to assign the value α representing the weight given to math expressions in the query. The first way is *fixed* ($\alpha = 0.5$) where math expressions and keywords weigh the same. The second way is *dynamic* defined as $\alpha = \frac{|E|}{|E|+|T|}$ for each query q, where E and T are the sets of all math expressions and keywords in q respectively.

In order to investigate if larger expressions should have heavier weights than smaller ones on queries with multiple math expressions, we evaluated two different ways to assign the individual weights w_e assigned to each math expression e on a given query q. The first way is *balanced*, where $w_e = \frac{1}{|E|}$ meaning that all math expressions weight the same independently of their size. The second way is *by size*, where larger expressions in the query are given heavier weights using $w_e = \frac{size(e)}{\sum_{x \in E} size(x)}$, where $size(x)$ returns the count of symbols in x.

We configured our system using 4 different settings using combinations of these 2 variables. We used these 4 settings for both the main arXiv tasks and the Wikipedia subtask. We fixed the similarity metric to **SM3** as defined in Section 3.3.2. The window size w was unbounded (all possible pairs) for the Wikipedia main subtask, and it was set to 2 for the arXiv main task because of memory limitations for larger index. Final submissions can be described as: *α-fixed w_e-by-size*, *α-fixed w_e-balanced*, *α-dynamic w_e-by-size*, *α-dynamic w_e-balanced*. Table 3.8 shows results using Precision@K for $K = \{5, 10, 15, 20\}$ on MathIR arXiv Main Task for relevant and partially-relevant matches. Table 3.9 shows the same evaluation metrics on optional MathIR Wikipedia Task. For space reasons, we have only included submissions from other systems containing highest scores for any category.

Note that a TREC evaluation tool was used to compute Precision@K for all participants. This tool's default behavior re-ranks documents with the same score based on their ids. Systems like ours use internal tie-breakers that were not submitted as part of the final score, and as a result Precision@K values were affected by re-ranking of documents. If we use the original ranks submitted, our best submissions obtain slightly higher precision values at all levels [136]. Complete precision values for all submission are available in the main NTCIR-12 MathIR competition paper [136].

In all result tables, we have included additional rows for "Ideal Pool Ranking" representing the

Table 3.8: NTCIR-12 MathIR arXiv Main Task. Precision@K Results using TREC evaluation tool

	Relevant				Partially Relevant			
Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
MCAT_allfields_lr_unif	0.2621	**0.2448**	0.2046	0.1810	**0.5586**	**0.5483**	0.5126	0.4707
MCAT_allfields_nowgt_unif	**0.2828**	0.2379	**0.2184**	**0.1948**	0.5448	0.5345	**0.5149**	**0.4897**
RITUW_ArXiv_run1 (α-*fixed* w_e-*by-size*)	0.2069	0.1517	0.1126	0.0948	0.4966	0.3966	0.3310	0.2879
RITUW_ArXiv_run2 (α-*fixed* w_e-*balanced*)	0.2069	0.1517	0.1126	0.0948	0.4966	0.3966	0.3310	0.2879
RITUW_ArXiv_run3 (α-*dynamic* w_e-*by-size*)	0.1379	0.1138	0.1034	0.0914	0.4897	0.4586	0.4207	0.3983
RITUW_ArXiv_run4 (α-*dynamic* w_e-*balanced*)	0.1379	0.1138	0.1034	0.0914	0.4897	0.4586	0.4207	0.3983
Ideal Pool Ranking	0.6966	0.5586	0.4644	0.4086	0.9655	0.9552	0.9172	0.8828

Table 3.9: NTCIR-12 optional MathIR Wikipedia Task. Precision@K Results using TREC evaluation tool

	Relevant				Partially Relevant			
Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
ICST_WikiMainTask	**0.4733**	**0.3767**	**0.2978**	**0.2617**	**0.8533**	**0.7900**	**0.7133**	**0.6600**
RITUW_wiki_run1 (α-*fixed* w_e-*by-size*)	0.2533	0.2367	0.2089	0.1983	0.4933	0.4833	0.4889	0.4733
RITUW_wiki_run2 (α-*fixed* w_e-*balanced*)	0.2533	0.2467	0.2156	0.2017	0.4933	0.4900	0.4822	0.4700
RITUW_wiki_run3 (α-*dynamic* w_e-*by-size*)	0.1600	0.1300	0.1244	0.1267	0.3867	0.3633	0.3733	0.3617
RITUW_wiki_run4 (α-*dynamic* w_e-*balanced*)	0.1600	0.1400	0.1311	0.1300	0.3800	0.3667	0.3644	0.3583
Ideal Pool Ranking	0.8400	0.6967	0.5956	0.5133	0.9467	0.9400	0.9289	0.9217

Table 3.10: NTCIR-12 optional MathIR Wikipedia Formula Browsing Task. Precision@K Results using TREC evaluation tool

	Relevant				Partially Relevant			
Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
MCAT-browse_allfields_nowgt_unif	**0.4900**	**0.3900**	**0.3317**	**0.2825**	**0.9100**	**0.8400**	**0.8067**	**0.7687**
RITUW-formula_run1 (**SM1**)	0.4150	0.3150	0.2650	0.2200	0.8100	0.7450	0.7117	0.6737
RITUW-formula_run2 (**SM2**)	0.4250	0.3175	0.2567	0.2200	0.8150	0.7550	0.7200	0.6938
RITUW-formula_run3 (**SM3**)	0.4400	0.3225	0.2700	0.2300	0.8400	0.7650	0.7317	0.7063
RITUW-formula_run4 (**SM4**)	0.4450	0.2925	0.2517	0.2200	0.8250	0.6825	0.6533	0.6100
Ideal Pool Ranking	0.7900	0.6400	0.5383	0.4725	1.0000	1.0000	0.9933	0.98

Table 3.11: NTCIR-12 optional MathIR Wikipedia Formula Browsing Task. Precision@K Results using submitted ranks

	Relevant				Partially Relevant			
Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
MCAT-browse_allfields_nowgt_unif	**0.5150**	**0.4050**	**0.3450**	**0.3000**	**0.9300**	**0.8650**	**0.8300**	**0.8012**
RITUW-formula_run1 (**SM1**)	0.4300	0.3400	0.2933	0.2450	0.8400	0.7800	0.7533	0.7225
RITUW-formula_run2 (**SM2**)	0.4450	0.3675	0.3100	0.2687	0.8550	0.8125	0.7833	0.7638
RITUW-formula_run3 (**SM3**)	0.4900	0.3750	0.3283	0.2812	0.8750	0.8175	0.7833	0.7563
RITUW-formula_run4 (**SM4**)	0.4900	0.3750	0.3217	0.2937	0.9000	0.8250	0.8033	0.7762
Re-ranking Upper Bound	0.7450	0.5625	0.4433	0.3700	1.0000	0.9925	0.9683	0.9375
Ideal Pool Ranking	0.7900	0.6400	0.5383	0.4725	1.0000	1.0000	0.9933	0.9800

performance of an ideal system with perfect relevance-based ranking for all results in each pool. This represents a soft upper bound for the current pool of results that does not consider additional relevant and partially-relevant documents/formulas in the collection that might be absent from each pool. In addition, we have included rows for "Re-ranking Upper Bound" in Tables 3.11 and 3.12 to represent the maximum Precision@K values that our system could get if the top-1000 candidates selected by the core are re-ranked according to their relevance scores.

Table 3.12: NTCIR-12 optional MathIR Wikipedia Formula Browsing Task. Precision@K Results using ranks for concrete and wildcard queries

		Relevant				Partially Relevant			
Query Type	Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
Concrete	RITUW-formula_run1 (**SM1**)	0.4800	0.3550	0.2900	0.2375	**0.9400**	**0.8850**	**0.8267**	**0.7950**
	RITUW-formula_run2 (**SM2**)	0.4200	0.3300	0.2667	0.2300	0.9200	0.8550	0.8000	0.7700
	RITUW-formula_run3 (**SM3**)	0.5200	0.3500	0.2933	0.2500	0.9100	0.8600	0.8133	0.7750
	RITUW-formula_run4 (**SM4**)	**0.5300**	**0.3700**	**0.3167**	**0.2775**	0.9100	0.8250	0.8067	0.7700
	Re-ranking Upper Bound	0.7200	0.5400	0.4167	0.3375	1.0000	1.0000	0.9800	0.9325
	Ideal Pool Ranking	0.7300	0.5800	0.4733	0.4000	1.0000	1.0000	0.9967	0.9800
Wildcard	RITUW-formula_run1 (**SM1**)	0.3800	0.3250	0.2967	0.2525	0.7400	0.6750	0.6800	0.6500
	RITUW-formula_run2 (**SM2**)	**0.4700**	**0.4050**	0.3533	0.3075	0.7900	0.7700	0.7667	0.7575
	RITUW-formula_run3 (**SM3**)	0.4600	0.4000	**0.3633**	**0.3125**	0.8400	0.7750	0.7533	0.7375
	RITUW-formula_run4 (**SM4**)	0.4500	0.3800	0.3267	0.3100	**0.8900**	**0.8250**	**0.8000**	**0.7825**
	Re-ranking Upper Bound	0.7700	0.5850	0.4700	0.4025	1.0000	0.9850	0.9567	0.9425
	Ideal Pool Ranking	0.8500	0.7000	0.6033	0.5450	1.0000	1.0000	0.9900	0.9800

In order to address our research questions regarding similarity between math expressions, we tested the 4 different metrics define in Section 3.3.2 for the optional MathIR Wikipedia Formula Browsing Task. Table 3.10 shows results using Precision@K for $K = \{5, 10, 15, 20\}$ on this task for relevant and partially-relevant matches. Table 3.11 shows the same evaluation when the submitted ranks are used without reranking ties by document name. For metrics **SM1**, **SM2** and **SM3**, w was unbounded (all pairs). Finally, Table 3.12 shows our Precision@K results for this task for relevant and partially-relevant matches divided between queries with and without wildcards. There were a total of 20 concrete queries (without wildcards) and 20 wildcard queries in the workload.

In terms of space, our system required 8,348.39 MB in hard drive to index all formulas in the arXiv collection and 580.51 MB for the Wikipedia dataset. These quantities require between 2.0 and 2.5 times that space when loaded into RAM. In terms of run time, we had the following (mean, minimum, maximum, median) times (in seconds): MathIR arXiv Main Task (27.54, 2.77, 178.51, 16.014) and optional MathIR Wikipedia Task (37.83, 1.33, 176.06, 33.84). In the case of the optional MathIR Wikipedia Formula Browsing Task, we obtained: M_1 (2.67, 0.10, 64.13, 1.07), M_2 (12.75, 0.17, 109.61, 3.61), M_3 (45.26, 0.58, 1032.39, 8.58), M_4 (29.80, 0.18, 718.70, 4.67). Note that these run times are typically longer for many queries containing wildcards. For example, M_4 requires (13.05, 1.26, 66.97, 4.50) for concrete queries, but (46.55, 0.18, 718.70, 4.82) for wildcard queries. We use a Ubuntu Linux 14.04 server with 24 Intel Xeon processors (2.93GHz) and 96GB of RAM. While some indexing operations were parallelized, *all retrieval times are reported for single-threaded processing.*

Discussion

For both the arXiv and Wikipedia main tasks, we generally obtained better precision using *α-fixed*. Most queries in the MathIR arXiv Main Task contained multiple keywords and at most one math expression making the *dynamic* setting give larger weights to keywords. In most cases, our system performed better when math expressions in the query had the same weight as keywords. A detailed analysis of results showed that this might be due to multiple cases where documents that only matched keywords were considered irrelevant. In particular, this happened often when these keywords were very generic terms like “define”, “name”, “arithmetic”. Also, certain keywords like “mean” are ambiguous and should only be matched in context. Our current search for keywords in text does not consider context, order or proximity of matched keywords, which results in many irrelevant documents matching only keywords ranked very high.

In terms of weights assigned to each math expression in queries with multiple expressions, we found that there was not much difference between setting $by - size$ and *balanced*. However, in the current query sets most queries contain only one math expression meaning that current data might not be enough to observe a difference if any exists. Analysis of queries with multiple expressions shows that, in many cases, a document is not considered relevant unless all math expressions are matched. The relevance of a math expression in a query is likely to depend on other factors.

Overall, our results show that simple linear combination of weights for keywords and math expressions is not enough for search, and that interactions between math expression and text in their context should be considered in the future.

We can compare our proposed similarity metrics using the results from the optional MathIR Wikipedia Formula Browsing Task shown in Tables 3.11. Note that for this analysis we only consider Precision@K produced by our ranks originally submitted. As expected, our core engine ranking (**SM1**) provides a baseline performance before re-ranking with slightly worse results than our re-ranking metrics. Overall, the Maximum Subtree Similarity (**SM4**) has the best performance of our four metrics. However, a detailed analysis separating concrete from wildcard queries shows that different metrics perform better under certain conditions (see Table 3.12).

For concrete queries, **SM4** performed best in terms of ranking the most relevant matches higher, but **SM1** was the best for partially relevant matches. **SM4** achieved better performance for relevant results because in many cases some expressions became exact matches after variable unification and were considered relevant by evaluators. In the case of disconnected exact sub-matches, our detailed matching algorithm enforces a connected component constraint and as a result some good candidates might be ranked low by the scoring functions **SM2**, **SM3** and **SM4** based on the size of the largest sub-match. However, disconnected exact sub-matches tend to be good only if they appear on similar baselines. For example, for the query $u + v + x + y + z$, the match $u + v - y + z$ might be considered acceptable, but $\frac{u+v}{y+z}$ might not. Also, **SM3** and **SM4** increase the rank of smaller unified matches that often were considered to be less relevant than disconnected exact matches.

For wildcard queries, all metrics using wildcard expansion (**SM2**, **SM3**, **SM4**) performed better on average than the core engine (**SM1**) that only considers single symbol matches. In terms of relevant matches, the gap between these three metrics that consider wildcard expansion is small. At the level of partial matches, **SM4** performs better than **SM3** because multiple slightly relevant unified matches are ranked higher. For queries containing polynomials, wildcard horizontal expansion to the left matched high degree polynomials that were considered irrelevant.

Unification of variables and constants was useful when applied over candidates with high structural match. However, multiple irrelevant expressions are ranked too highly when unification is applied over partial structural matches.

By comparing the best results for each metric (**SM1-SM4**) to the "Re-ranking Upper Bound" and "Ideal Pool Ranking" in Tables 3.11 and 3.12, we can measure how much room there is for improvement both in terms of initial candidate selection and re-ranking. For Precision@20, the gap between "Ideal Pool Ranking" and "Re-ranking Upper Bound" is currently 10.25% for relevant results meaning that, in average, there are at least 2 out of 20 relevant results in the ideal pool that are not selected by our system as initial candidates. For concrete queries only, this gap is just 6.25% (around 1 candidate), and for wildcard queries the gap is 14.50% (around 3 candidates). In terms of re-ranking, the gap for Precision@20 for relevant results between the best re-ranking metric and the current upper bound 7.63% overall, 6.00% for concrete-only and 9% for wildcard-only. The larger performance gaps are currently on queries containing wildcards. Note that the values for our "Re-ranking Upper Bound" are higher than the best submission (7% of difference in Precision@20 for relevant results), meaning that we could achieve similar precision values by improving our re-ranking functions using the same candidates from the core.

3.4.3 Experiment Set 3: Combining SLT and OPT in a 3-layer Model

In this section, we describe the set of experiments performed to evaluate the complete formula retrieval model (Tangent-S) using three layers. We also make comparisons of performance using isolated and combined SLT and OPT representations.

Similar to some of the experiments described in Section 3.4.2, here we also use data from the NTCIR-12 MathIR competition [136]. Specifically, we use data from the optional MathIR Wikipedia Formula Browsing Task which has a corpus of 319,689 articles from English Wikipedia with more than half a million formulae, and defines 40 queries for isolated formula retrieval: 20 are concrete (without wildcards) and 20 include wildcards.

For SLTs, matching works mostly as defined for Tangent-3.1 [27], except that we now restrict unification to occur between single character identifiers, or within identifiers with two or more characters, but not between these two groups. This mitigates the issue of spurious unification matches between variables and functions leading to bad candidates being ranked too high described in Section 3.4.2.

For re-ranking, we use a variation of **SM4** based on Maximum Subtree Similarity (MSS), where to better support linear regression, we replaced the negative counts by unified precision and recall without unification (**SM5**), producing a similar lexicographic ordering as before using values in the range $[0, 1]$. The recall and precision computed here differ from those used by other formula retrieval methods, in that they are computed from subtrees after matching constraints are enforced.

At NTCIR-12, the top-20 results for each topic from 8 submissions were evaluated for relevance. Each result was assessed by two human evaluators who scored them from 0 (irrelevant) to 2 (relevant). These scores were combined and each formula has a final relevance score between 0 and 4. A total of 2687 relevance assessments were produced by this method.

We evaluated the ranks produced by the model for each representation (SLT, OPT) at each retrieval stage (*Core, Matching, Regression*). Core is the same ranking produced by **SM1** while Matching is the ranking produced by **SM5**. The combined SLT/OPT approach (Section 3.3.4) is also considered, for a total of seven conditions. At the first stage, we select the top 1000 candidates for each query using $w = All$ (all tuples) and $EOB = True$. Given the limited number of relevance assessments available, for conditions using Linear Regression we grouped each concrete query with its corresponding wildcard version, and created 20 data folds. We then used cross-validation, repeatedly using one fold to test and the remaining 19 folds to train a linear regressor, until all queries have been processed.

Table 3.13 compares our three-stage ranking systems, and systems participating in the NTCIR-12 competition. We used the TREC eval tool to compute the values of Precision@K with K in $\{5, 10, 15, 20\}$. Unlike the Tangent-3 submissions at NTCIR-12 [27], we ensured that the lexicographic order used for ranking after structural matching (the *Matching* conditions) was preserved by the TREC eval tool, by computing the ranks produced by the fixed-order MSS and tie breaker scores, and then using the reciprocal rank $(1/r)$ of each score vector as the final score. Formulae with identical score vectors are re-ranked by the TREC eval tool based on document id.

Mainly because of a large number of unrated formulas that are unfairly assumed to be irrelevant, Precision@K scores for the proposed conditions, specially the OPT-based, are lower than those for

Table 3.13: NTCIR-12 optional MathIR Wikipedia Formula Browsing Task. Average Precision@K per topic

	Relevant				Partially Relevant			
Method	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
MCAT [51]	**0.4900**	**0.3900**	**0.3317**	**0.2825**	**0.9100**	**0.8400**	**0.8067**	**0.7687**
Tangent-3 [27]								
Core	0.4150	0.3150	0.2650	0.2200	0.8100	0.7450	0.7117	0.6737
Matching	0.4450	0.2925	0.2517	0.2200	0.8250	0.6825	0.6533	0.6100
SLT								
Core	0.4000	0.3025	0.2567	0.2125	0.7900	0.7275	0.6950	0.6562
Matching	0.4550	0.3450	0.2817	0.2462	0.8350	0.7725	0.7400	0.6913
Regression	0.3900	0.2900	0.2450	0.2000	0.6300	0.5525	0.5117	0.4675
OPT								
Core	0.3650	0.2550	0.2050	0.1700	0.6250	0.4825	0.4200	0.3662
Matching	0.3550	0.2475	0.2017	0.1787	0.5550	0.4400	0.3800	0.3425
Regression	0.3150	0.2475	0.2000	0.1700	0.6250	0.4975	0.4317	0.3850
Combined	0.4400	0.3150	0.2583	0.2162	0.7000	0.6075	0.5550	0.5112

Table 3.14: NTCIR-12 optional MathIR Wikipedia Formula Browsing Task. Average Bpref per topic.

	Core		Matching		Regression		
Matches	SLT	OPT	SLT	OPT	SLT	OPT	Comb.
Relevant	0.4207	0.4227	0.4786	0.4760	0.5240	0.5127	**0.5530**
Partially Relevant	0.5126	0.4241	0.5351	0.4206	0.5569	0.5492	**0.5620**

Table 3.15: Average nDCG@K of ranks per topic for judged NTCIR-12 Wikipedia Formulae.

	All Topics		Concrete Only		Wildcard Only	
Condition	@5	@20	@5	@20	@5	@20
SLT						
Core	0.7109	0.7002	0.7991	0.7727	0.6236	0.6277
Matching	0.7534	0.7218	0.8033	0.7776	0.7036	0.6659
Regression	0.7943	0.7723	0.8031	0.7958	0.7855	0.7488
OPT						
Core	0.6978	0.7184	0.7889	0.7891	0.6066	0.6478
Matching	0.7459	0.7446	0.7889	0.8018	0.7028	0.6874
Regression	0.7519	0.7331	0.8008	0.7773	0.7031	0.6888
Combined	**0.8136**	**0.7908**	**0.8131**	**0.8088**	**0.8141**	**0.7728**

systems in the competition, with the exception of SLT Matching which are higher than Tangent-3 Matching. However, Tangent-S is simpler and faster than MCAT [27, 51], the best performing system in the competition.

The binary preference (Bpref) metric ignores unrated matches, and quantifies the ability of the ranking method to keep judged relevant matches ranked higher than irrelevant ones. For a given query with a number of relevant document R and irrelevant documents N, we define $NonRelevant(r)$ as the number of irrelevant elements in a ranking that are listed above the relevant match r. We then defined Bpref as follows:

$$Bpref = \frac{1}{R}\sum_{r}\left[1 - \frac{min(NonRelevant(r), R)}{Min(N, R)}\right]$$

Table 3.14 shows a comparison of Bpref values across different conditions of our own method. Unfortunately, Bpref values are not available for the original systems in the competition.

In terms of Bpref, we can see that in most cases the values increase at each layer of the retrieval model. For partially relevant results, with SLTs Bpref goes from 0.4207 in the initial set

Table 3.16: Top-5 formula results for Query-12 for each ranking stage for each representation

Query-12: $\mathcal{O}(mn\log m)$

	Symbol Layout Trees			
	Core	**Matching**	**Regression**	
1.	$\mathcal{O}(mn\log m)$	$\mathcal{O}(mn\log m)$	$\mathcal{O}(mn\log m)$	
2.	$\mathcal{O}(mn)$	$\mathcal{O}(VE\log V)$	$\mathcal{O}(mn\log p) = \mathcal{O}(n\log n)$	
3.	$\mathcal{O}(mnp)$	$\mathcal{O}(VE\log V\log(VC))$	$\mathcal{O}(mn) = \mathcal{O}\left(n^3\log n\right)$	
4.	$\mathcal{O}(m+\log n)$	$\mathcal{O}(Tm) = \mathcal{O}\left(n^2 m\log n\right)$	$\mathcal{O}\left(d^5 n\log^3 B\right)$	
5.	$\mathcal{O}\left(m\sqrt{n}\log n\right)$	$\mathcal{O}(nk\log k)$	$\mathcal{O}\left(mnr^2\log\frac{1}{\epsilon}\right)$	
	Operator Trees			Combined
	Core	**Matching**	**Regression**	**Regression**
1	$\mathcal{O}(mn\log m)$	$\mathcal{O}(mn\log m)$	$\mathcal{O}(mn\log m)$	$\mathcal{O}(mn\log m)$
2	$\mathcal{O}(mn)$	$\mathcal{O}(nk\log(n))$	$\mathcal{O}(mn\log p) = \mathcal{O}(n\log n)$	$\mathcal{O}(mn\log p) = \mathcal{O}(n\log n)$
3	$\mathcal{O}(mnp)$	$\mathcal{O}(VE\log V)$	$\mathcal{O}\left(mnr^2\log\frac{1}{\epsilon}\right)$	$\mathcal{O}\left(M(m)\log^2 m\right) = \mathcal{O}(M(n)\log n)$
4	$\times(mnp)$	$\mathcal{O}(n\log n)$	$\mathcal{O}(pn(m+n\log n))$	$\mathcal{O}\left(mnr^2\log\frac{1}{\epsilon}\right)$
5	$\mathcal{O}(mr)$	$\mathcal{O}(nk\log k)$	$\mathcal{O}(Tm) = \mathcal{O}\left(n^2 m\log n\right)$	$\mathcal{O}(mn) = \mathcal{O}\left(n^3\log n\right)$

of candidates to 0.5240 after using linear regression, and from 0.4227 to 0.5127 for OPTs. When SLT and OPT scores are combined, Bpref increases to 0.5530. This confirms that each step of the pipeline improves the quality of the produced ranks, and that combining representations helps.

As the number of expressions judged per topic is relatively small, and given that our new conditions produce many unrated results in the top-20, we have used a different approach to further analyze the rankings produced. We re-ranked all judged formulas for each topic using each stage of our retrieval model, and we compared these ranks against the ideal rankings using nDCG@K with K $= \{5, 20\}$ as shown in Table 3.15. Consistently, the *Matching* stage improves the ranking quality produced by the initial spectral symbol pair matching for both representations. While linear regression consistently increases nDCG for SLTs, it produces a small decrease for OPTs compared to the lexicographic order of the same scores from the *Matching* stage. This may be due to a greater collinearity between Recall of nodes and MSS in the OPT space. In general, nDCG values for Concrete topics are higher than for the Wildcard topics. In the current matching procedure, we have observed that allowing Wildcards to match subtrees in the presence of poor unifications can cause bad partial matches to be ranked high.

An analysis of relevance against similarity scores reveals that while the means of MSS and node recall increase with the relevance of judged matches on both representations, node precision after unification is not well correlated with relevance and might hurt linear regression predictions and even re-ranking in general. This is not surprising since some bad partial matches have low recall but high precision. For example, a single query wildcard can be matched to an entire arbitrary expression tree with perfect precision.

Table 3.16 illustrates the differences in the Top-5 ranks for each stage of the model for the query $\mathcal{O}(mn\log m)$, for both SLTs and OPTs. Differences in structure for each representation change the initial set of candidates extracted from the collection. The *Matching* columns show how the unification process helps in increasing the rank of partial matches that become exact after unification. This query illustrates how linear regression can sometimes produce less intuitive rankings than the simpler lexicographic match score ordering. It also shows how the OPT representation can give better rankings to equivalent expressions that have a slightly different layout like the candidate $\mathcal{O}(nk\log(n))$ after unification.

It is important to acknowledge noise in the NTCIR-12 formula data. Many expressions are incorrectly but consistently converted into Content MathML from LaTeX. For example, the sub expression $f(x)$ is almost always converted to the tree corresponding to $f \times x$. Such errors in the source can lead to many undesirable partial matches for OPTs at retrieval time.

3.5 Summary

We have presented Maximum Subtree Similarity (MSS), a new metric for similarity in mathematical formulae, along with an efficient and effective three-stage cascaded retrieval model for formula retrieval. In the first stage, the symbol pair retrieval model out-performs earlier symbol pair-based formula retrieval systems, being more efficient in space and time. The second stage re-ranks the top-k matches by MSS and produces state-of-the-art results for the NTCIR-11 Wikipedia formula retrieval task. Similarity ratings from human participants agree substantially with formula rankings produced using MSS. Finally, the third layer helps to further approximate this perceived similarity between formulas if training data is available.

We have evaluated the performance of our proposed approach using multiple MathIR tasks from NTCIR-11 and NTCIR-12. Our research questions have been partially answered. In terms of weighting query terms, we need a better model for weight distribution that considers interactions between math expressions and text. Some query terms might be considered more important based on their intrinsic value to the query topic independently of their size in terms of count of symbols.

The optional MathIR Wikipedia Formula Browsing Task in particular was useful to provide additional answers to our questions. Our current set of candidates obtained by the core engine using the Dice coefficient (**SM1**) is competitive, but there is considerable room for improvement. A detailed matching algorithm was generally helpful to rank the most relevant matches higher, but some partially-relevant disconnected matches are ranked too low. A less constrained matching algorithm can be helpful in ranking these candidates higher. On the other hand, the unification process still needs to be further constrained to avoid ranking many irrelevant partial matches too high. Finally, both the Dice Coefficient and the MSS performed better under different circumstances. Further analysis can be used to create similarity metrics that perform better.

We also presented a comparison of the performance for two math expression representations using a three-layer retrieval model. We proposed a simple way to combine Symbol Layout Tree (SLT) and Operator Tree (OPT) representations into a single retrieval model. Overall, this combined model produced better rankings than the individual representations. In this study, we only considered original Tangent-3 scores for the final linear regression step. However, the method proposed here may be used to incorporate and study additional similarity scores that may be better predictors for relevance (e.g., symbol and structure recall before unification). Similarity scores can also be combined with non-linear models for more accurate candidate selection and relevance prediction, while maintaining fast retrieval.

For future work, larger-scale human experiments are needed to identify which features affect the perception of formula similarity, and how different populations identify formulae as similar (e.g., for mathematical experts vs. non-experts). Retrieval efficiency may be improved by compressing dictionaries and postings lists, and by using an implementation of weak-AND [13] or a more fine-grained implementation of max-score [121]. Retrieval effectiveness can be improved by implementing more of the query functionality in the engine, improved methods for ranking documents based on multiple partial matches (whether from one or from multiple query expressions), and integrating a more sophisticated model for keyword search that considers relationships between mathematical entities and formulas based on analysis of the context in which the math formulas are found.

Chapter 4

Lecture Video Summarization

Many lecture recordings are available online and they represent a very useful resource for many students from different levels. Consider the case of a linear algebra student who wants to find the particular portion of a lecture where the professor explains identity matrices. This portion might be just 5 minutes long within a one hour long video. In this case, it will be probably faster to manually find the portion in a short summary than in the raw video.

Summaries of lecture videos can be useful tools for user video navigation, and also for automated indexing and retrieval of lectures as we will show in Chapter 6. For multiple fields of study, like mathematics or physics, in which explanations are usually given using handwritten notes and graphics on the whiteboard or chalkboard, producing these summaries manually is time consuming and requires detailed annotations of the handwritten content. However, many lecture videos in these fields are only annotated using high level tags describing the topics covered. Sometimes, audio transcripts are available but these do not always describe the entire whiteboard/blackboard contents. If specialized hardware for raw trace capture (e.g. an interactive whiteboard) is available at lecture recording time, then the traces could be recorded and online symbol recognition methods can be use to recognize them later. However, in the absence of human annotations and specialized hardware, particularly for pre-existing collections of lecture recordings, we require robust automated methods that are able to extract and summarize such whiteboard contents from the video itself.

Video summarization for slide-based lecture videos has been studied in the past [54]. In this paper, we focus on providing a method for extraction and summarization of handwritten whiteboard content. Existing methods for lecture video summarization typically focus on local detection of changes in whiteboard/blackboard/slide content [20, 54]. Such changes are associated with slide transitions or whiteboard/blackboard erasing events. The problem with such approaches is they rely on detecting sharp changes in a function of time representing the content. Sometimes, if the change is not sharp enough, a transition/erasing event will not be detected, resulting in under-segmentation. In videos using whiteboard or blackboard these changes can happen gradually and can be missed if small detection windows are used.

We propose a divide-and-conquer segmentation method that starts by analyzing conflicting regions of content at a global scale, and recursively splits the video into units that contain few or no content conflicts. Two connected components are in conflict if there is an overlap in the space they occupy on the whiteboard but they exist during different time intervals. This happens when the whiteboard gets erased and something new is written on the same region. The proposed segmentation allows the summarization of the video using a small set of key-frames. A single key-frame might combine portions of content that never co-existed on the whiteboard, but that

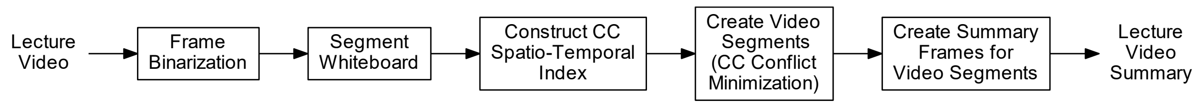

Lecture Video
Frame Binarization
Segment Whiteboard
Construct CC Spatio-Temporal Index
Create Video Segments (CC Conflict Minimization)
Create Summary Frames for Video Segments
Lecture Video Summary

Figure 4.1: Lecture Video Summarization Architecture

occupied different spaces and can be displayed on a single image for better compression rates, producing shorter summaries. For example, a 45 minute-long video might be summarized in just 10 frames.

In this paper we explore the following research questions: **Q1.** How can we reliably extract whiteboard contents from lecture videos? **Q2.** How can we produce a static summary for a lecture video using a minimal number of frames that contain all handwritten content of the lecture?

Contributions. We propose a novel method for lecture video summarization based on minimization of conflicts between content regions. Second, a spatio-temporal index that can be used to navigate lecture videos based on the handwritten content on the whiteboard. Finally, given the lack of labeled data for this particular domain, we provide a small dataset of labeled lecture videos that can be used for training and testing of newer approaches for this and related problems. Our dataset and tools for ground truth generation will be publicly available[1].

4.1 Methodology

We propose a method (see Figure 4.1) that given a lecture video is able to produce a small set of frames containing the handwritten content from the whiteboard. As illustrated in Figure 4.2, our method is designed for videos recorded using a single still camera in a classroom. We assume that the whiteboard will be the largest object in the image surrounded by some background objects. Some moving elements like the instructor will be present on the video as well.

Frame Sampling and Binarization. We sample frames from the video at a sampling rate FPS (frames per second) to obtain a subset F. We have determined empirically that $FPS = 1$ is enough to capture relevant changes. Each selected frame is then preprocessed for background estimation and removal, and then binarized using a combination of two methods: a high-recall, low-precision machine learning binarizer, and low-recall high-precision Otsu's [87] binarization. An example of this procedure is shown in Figure 4.3.

Global thresholding using methods like Otsu's [87] do not work well in gray scale image for this application. A single global threshold is unable to separate the whiteboard content from the background pixels due to non-uniform illumination on the board. Adaptive threshold methods do not work well because of many false positives in large empty regions of the whiteboard. In addition, dirty whiteboards and old markers produce traces that are hard to distinguish from background even for humans. To deal with these issues, we use a background subtraction method to generate edge frame images F^E as shown in Figure 4.3c.

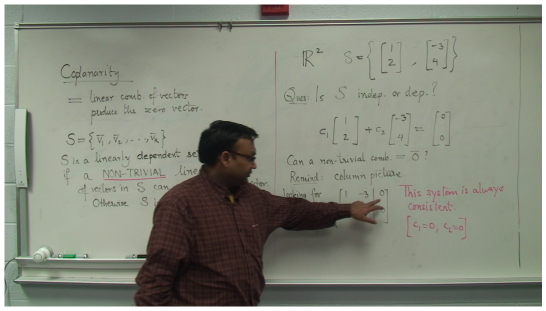
For a given frame f, we first apply a bilateral filter [117] with Sigma color $B_{sc} = 13.5$ and sigma space $B_{sp} = 4.0$ for smoothing the whiteboard background while preserving the handwriting edges. Then, we estimate the background using a median blur filter with aperture size $B_{blur} = 33$. We then subtract estimated background from the smoothed image to obtain the difference on each RGB channel. The raw difference includes positive and negative values depending on which pixels

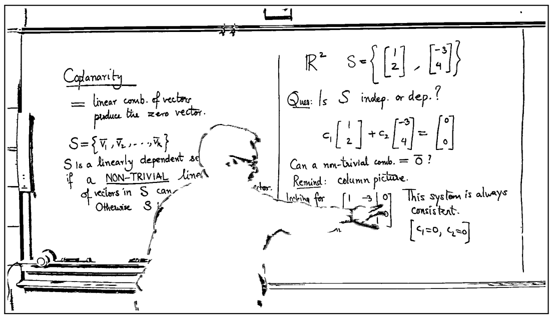
[1]`https://cs.rit.edu/~dprl/Software.html`

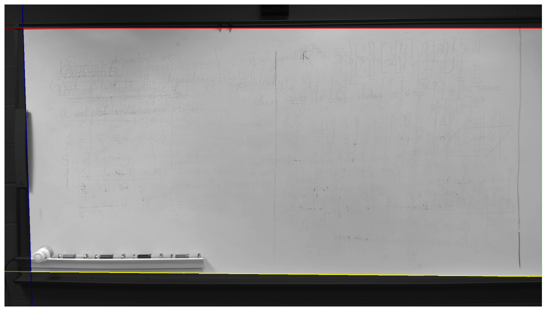
are lighter or darker than the estimated background. Since we work with whiteboards, we only need the pixels darker than the background (negative values). We make all positive values equal to zero and then we change the sign of the negative values. We finally combine the differences on the three channels into a single edge image f^e by keeping the maximum value of all channels per pixel.

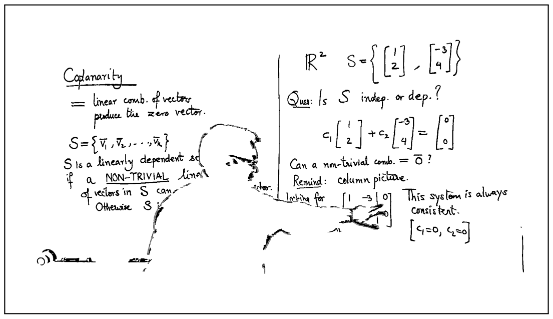
Note that for chalkboards/blackboards, we would simply need to make the negative values in the raw difference image equal to zero, and the rest of the binarization procedure would remain mostly unchanged.

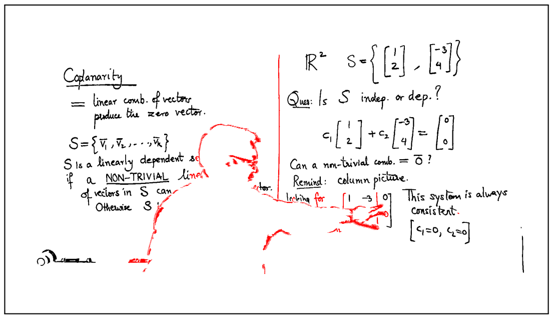
We use a Random Forest classifier [12] for window-based pixel-level binarization because they produce reasonably accurate classification results in faster times than other machine learning techniques. We train it using patches of size $T_w \times T_w$ ($T_W = 7$) randomly sampled from fully labeled binary key-frames in training data. The goal is to learn the class of the pixel at the center of each window given contextual information in the edge space. We bias the patch sample by forcing a pro-


(a) Raw Frame


(b) Binarization


(c) Whiteboard Detection


(d) Whiteboard Segmentation


(e) CC Stability Analysis

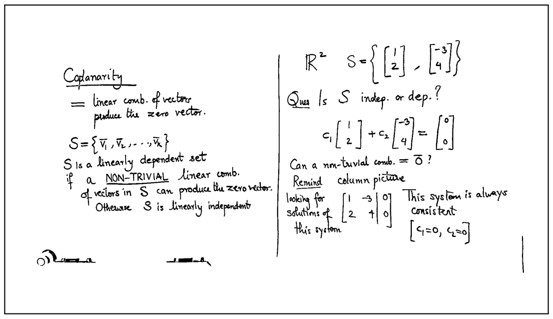
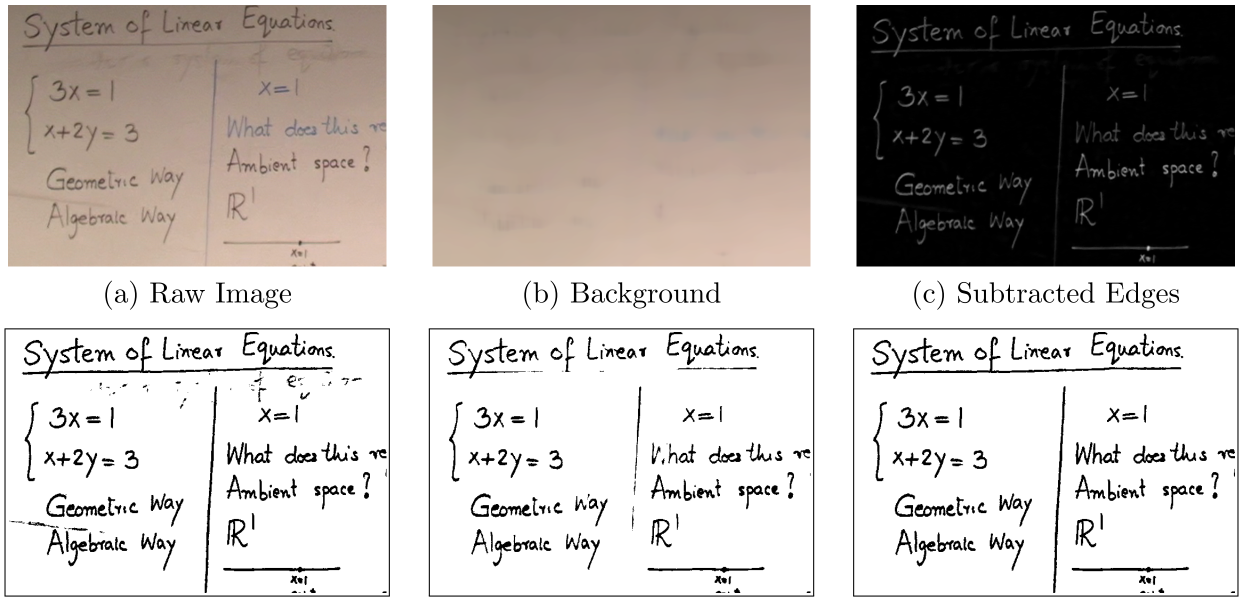

(f) Reconstructed Frame

Figure 4.2: Overview of intermediate steps of our video summarization approach.



(a) Raw Image (b) Background (c) Subtracted Edges

(d) RF Binary (e) Otsu Binary (f) Hysteresis

Figure 4.3: Overview of our binarization process.

portion of $T_{fg} = 50\%$ patches to be taken from foreground elements, where T_{fg} is typically higher than the actual proportion of foreground pixels on the whiteboard. Using object labels provided in the ground truth, we only sample patches from whiteboard pixels as we assume that general background and the speaker will be removed from binary images using other processes later. For the whiteboard background pixels sampled, we bias their distribution by assigning each pixel a probability proportional to their intensity in the edge frame space, adding 1 to all intensities to smooth probabilities for zero intensity pixels. In this sense, the harder classification cases of the strong edges that are not handwriting or that are close to its boundaries will be well represented in our random sample. After sampling the training patches, we train our Random Forest classifier [12] using $T_{trees} = 16$ trees, maximum tree depth of $T_{depth} = 12$ and maximum number of features to consider at each split of $T_{feat} = 32$.

We finally binarize each frame edge image f^e using hysteresis, similar to Canny edge detection, by combining a weak (high recall, low precision) with a strong (low recall, high precision) binary image. The weak image is obtained using the window-based pixel-level Random Forest classifier described before (See Figure 4.3d). The strong image is obtained using Otsu's [87] binarization (See Figure 4.3e). The final binary image f^b is generated by overlapping the two images and keeping all connected components (CC) from the weak image that have at least one pixel overlap in the strong image.

We use a combined approach because the Random Forest generally produces easy to read binary traces and can even recover trace pixels that are hard to separate from the background in the original image (See Figure 4.3d), but it also produces false positives. On the other hand, Otsu's binarization [87] produces few false positives but many broken traces. The final binary keeps most handwriting CCs and will have most noisy CCs removed (See Figure 4.3f).

Whiteboard Segmentation This procedure estimates the whiteboard region and removes from the binary frames any CCs that are not fully contained in this region. (See Figure 4.2c and

4.2d). Similar to the optimization approach used by Li et al. [54], we estimate the whiteboard region using two high confidence estimates, one for handwriting pixels and the other for general background pixels, to choose the best quadrilateral from a set of candidates. We use a coarse sample of frames (one every 10 seconds) to estimate handwriting and general background locations based on two pixel-wise temporal statistics: median and standard deviation.

The pixel-wise temporal median image captures the temporal background of the video by keeping the most stable objects (usually background) while removing unstable elements like the speaker and most handwriting. These properties make this image ideal for estimation of the boundaries of the whiteboard region and locations of background pixels. We apply Canny edge detection (Low threshold = 30, high threshold = 50) to this image to obtain a high confidence estimation of background pixels. Next, we use the Hough transform (radius resolution = 1 pixel, angular resolution = 1 degree, minimum line length = 100, maximum line gap = 10, minimum line intersections = 50) on the edge image to obtain candidate edges for the whiteboard region. We classify these edges based their angle and relative location as: top, bottom, left or right. We ignore diagonal lines that are more than 15 degrees away from the closest axis. To the list of candidate edges we add the boundaries of the image in case that no edge is detected on a particular direction.

The pixel-wise temporal standard deviation image captures what pixels change the most making it good for estimation of handwriting locations. We transform all sampled images to the same edge space used for binarization. We apply a temporal median blur filter using a window of size $C_{blur} = 11$ frames. This produces a set of edge frames where fast moving elements like the speaker are removed. From this set, we compute the final pixel-wise temporal standard deviation image. In this image, handwriting pixels tend to have the highest intensities. We threshold by setting a minimum intensity $C_{fg}^{\perp} = 5$, assuming that all pixels with intensity greater than or equal to $C_{fg}^{\perp}$ are very likely to be handwriting pixels.

Then we choose the quadrilateral whiteboard region using Hough transform line candidate edges and the estimates of handwriting and background pixels. We exhaustively evaluate all possible combinations of top, bottom, left and right edges to find a region that maximizes the C_{f1} criterion:

$$C_{f1} = \frac{2C_{fg}^{rec}C_{bg}^{rec}}{C_{fg}^{rec} + C_{bg}^{rec}} \times W_{area} \tag{4.1}$$

Where C_{f1} represent the harmonic mean of the high confidence foreground pixel recall C_{fg}^{rec} and high confidence background pixel recall C_{bg}^{rec} scaled by the area of the candidate W_{area}. The goal is to maximize both the proportion of assumed to be foreground pixels inside of the region and the proportion of assumed to be background outside of the region. We multiply by W_{area} to prefer larger candidate regions when many of them have very similar harmonic means. In some cases, this criteria might prefer unnecessary larger but safer regions that preserve recall with some loss in precision.

The final background removal step on a given frame f^b simply removes any CC that is not fully contained in the final whiteboard region. Applying this to every frame in F^B we obtain the frame set F^C where the background has been mostly removed and most CCs belong to either handwriting or foreground elements like the speaker.

CC Stability Analysis. We identify and group CCs that are stable for some interval of time in the video. The output of this analysis is a spatio-temporal index of groups of stable CCs. For each group, the index stores a timeline describing the life span of the group split into time intervals based on the original addition/deletion times of each stable CC in the group. For each time interval, the spatio-temporal index stores a refined image of the group of CCs and their location in the frame. We

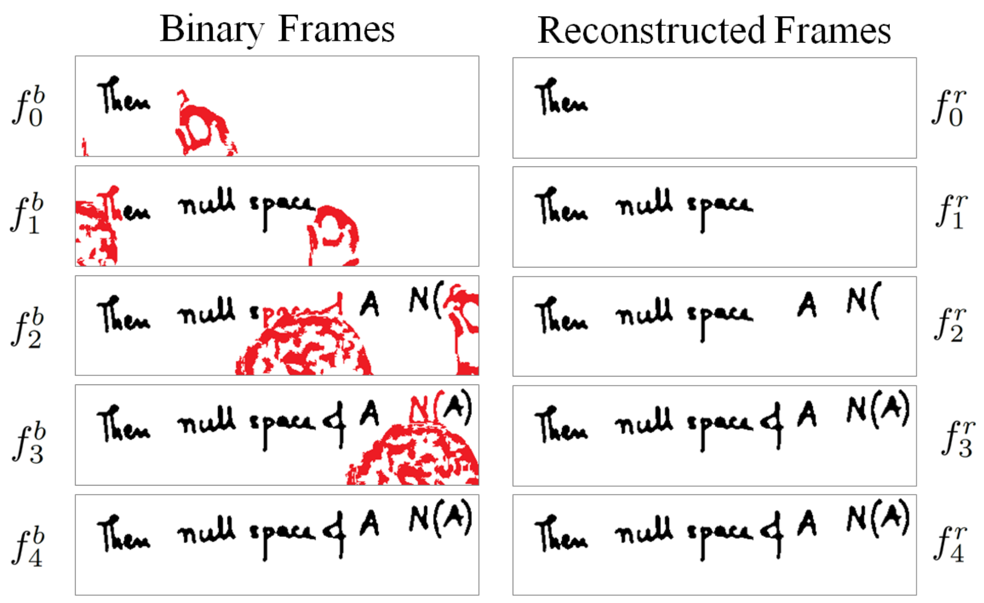


Figure 4.4: CC Stability Analysis. Stable CCs are shown in black while unstable CCs are shown in red. The reconstruction uses stable CCs to fill gaps lefts by removed unstable CCs

can then use this index to reconstruct the binary frames without moving objects like the speaker. Also, we can use it to detect conflicts between regions of content for temporal video segmentation.

Figure 4.4 shows a comparison between raw binary frames containing both stable and unstable elements and corresponding reconstructed binary frames containing only stable elements. Static elements in the background and handwritten content in the whiteboard are expected to produce CCs that can be matched between neighboring frames. Moving elements like the speaker are expected to produced CCs that are hard to match across frames becoming quite unique and easy to identify. Also, a moving speaker will block portions of the whiteboard causing stable CC to change in shape, merge and/or split for some frames. However, stable CCs remain with roughly the same shape on frames where they are not being occluded by the speaker.

The proposed method of analyzing stable CCs can be divided into two main steps: Stable CC identification, and Stable CC grouping.

Step 1: Stable CC identification. This step detects instances of unique CCs that appear in different frames within certain intervals of time. The input is the set of binary frames after background removal F^C. The output is a set of unique stable CCs S_{cc}. Given two binary frames f_i^c and f_j^c, we want to detect whether the CCs u and v, $u \in f_i^c$ and $v \in f_j^c$, might represent the same unique object in the video. We use temporal and spatial criteria for this task.

The spatial criterion is defined by computing the pixel-wise overlap between u and v. For this task, we use:

$$Space\,(u, v) = \frac{|p_u \cap p_v|}{|p_u|} \geq S_{space}^{\perp} \wedge \frac{|p_u \cap p_v|}{|p_v|} \geq S_{space}^{\perp} \tag{4.2}$$

where p_u and p_v represent sets of pixel locations of u and v respectively. We want the spatial overlap $|p_u \cap p_v|$ to represent at least $S_{space}^{\perp} = 92.5\%$ of both u and v pixels.

The temporal criterion is defined by the time difference between f_i^c and f_j^c. If t_i and t_j are the timestamps of f_i^c and f_j^c respectively, we define the temporal criterion as:

$$Temporal\left(t_i, t_j\right) = |t_i - t_j| \leq S_{time}^{\top} \tag{4.3}$$

Where the threshold $S_{time}^{\top} = 85$ seconds is the maximum time gap between f_i^c and f_j^c in order to consider u and v the same stable CC. These rules are applied between all pairs of frames on a moving window of length $S_{time}^{\top}$, and while for u and v the criteria might not hold directly, there might another intermediate CC w, $w \in f_k^c$ with timestamp t_k such that $Space\left(u, w\right) \wedge Space\left(w, v\right) \wedge Temporal\left(t_i, t_k\right) \wedge Temporal\left(t_k, t_j\right)$ and thus we would conclude that u, v and w are 3 instances of the same CC.

The next process is to take the set of unique CCs U_{cc} and identify the subset of stable S_{cc} believed to represent handwriting assuming that all background has been correctly removed previously. For this task, we use a threshold $S_{count} = 3$ representing the minimum number of frames in which a given unique CC must appear in order to be considered stable: $S_{cc} = \{u \in U_{cc} | Count(u) \geq S_{count}\}$. By applying this filter, we manage to remove CCs belonging to moving objects like the speaker. However, this filter might also eliminate handwritten content appearing for shorts periods of time. In a sense, S_{count} is related to the minimum time that a given element must remain on the whiteboard to be considered relevant. Unstable elements are sometimes noise or even mistakes made by the writer who might quickly erased them from the whiteboard. Speakers might not be completely removed from the video if they do not move for many seconds.

Step 2: Stable CC Grouping. This procedure takes as input the set of stable CCs, S_{cc}, groups them by overlaps in space and time and finally produces a spatio-temporal index of these groups. First, we identify the sets O_s and O_t as the sets of CCs that overlap in space and time respectively. If we define $[t_0^u, t_n^u]$ and $[t_0^v, t_m^v]$ as the time intervals where u and v appear respectively, then we can define $Overlap(u, v)$ as:

$$Overlap\left(u, v\right) = t_0^u - S_w < t_m^v \wedge t_0^v < t_n^u + S_w \tag{4.4}$$

$$O_s = \{(u, v) \in S_{cc} \times S_{cc} |\, |p_u \cap p_v| > 0 \wedge u \neq v\} \tag{4.5}$$

$$O_t = \{(u, v) \in S_{cc} \times S_{cc} | Overlap(u, v) \wedge u \neq v\} \tag{4.6}$$

Where $S_w = 5$ seconds is a small window allowing stable CCs separated by a small time gap to be considered as overlapping in time. We finally define the set $O_{ts} = O_t \cap O_s$ as the set of all stable CCs overlapping both in space and time.

Now, the sets S_{cc} and O_{ts} are used to compute G_{cc}, a partition of S_{cc} that groups all stable CCs having spatial and temporal overlaps. This is an attempt to group together different CCs that might represent a unique set of objects in the video. For example, a typical CC representing handwriting might suffer multiple gradual changes as it gets written, overwritten or partially erased. Grouping CCs also accounts for some minor binarization errors that consistently split/merge CCs of a set of content objects. Here all these related CCs are merged and treated as single units that change shape over time.

We start with $G_{cc} = \{\{u\} | u \in S_{cc}\}$. Then, we use every pair in O_{ts} to agglomerate the subsets in G_{cc}. We then compute the relevant time stamps for every group of CC in G_{cc} and create our spatio-temporal index to represent the extracted content from the whiteboard in terms of groups of CC and their corresponding timelines.

Frame Reconstruction. We use the spatio-temporal index to regenerate the binary frames of the video. First, we create refined images of each group of CCs in G_{cc} for each time interval in its timeline. These are images are pixel-wise averages of foreground pixels of CCs from the group that co-existed in that particular interval of time. We then use these refined images to create the reconstructed binary frames F^R.

Using this procedure we also fill the gaps in the video where stable CCs were not visible due to occlusion by foreground elements like the speaker. Refined images of groups will be placed in all frames within its timeline, filing many holes left behind by unstable CCs removed in the previous steps. Figure 4.4 shows an example of this procedure. In the original frames, the head and hand of the speaker block some content that is now visible in the reconstructed frames.

Segmentation Through Conflict Minimization This process creates a temporal segmentation P^R of the video by identifying and minimizing conflicts. Here, we define two regions of contents to be in conflict with each other if they occupy the same space in the whiteboard, but exist at different time intervals. Content that gets erased will be in conflict with anything that gets written on the same space. The goal of the proposed algorithm is to find suitable split points that will automatically separate such conflicting content into different video segments using a minimum number of splits. This also means that content that never co-existed in the whiteboard might be located on the same video segment as long as they are written on different whiteboard regions. Two main steps are required for this process: CC conflict detection and CC conflict minimization.

Step 1: CC Conflict Detection During the CC stability analysis, two sets of pairs of stable CCs with spatial overlap, O_s and O_{ts}, were identified. We define the set of conflicting pairs of CCs O_{conf} using $O_{conf} = O_s - O_{ts}$. In other words, O_{conf} is the set of pairs of CCs that overlap in space but do not overlap in time.

Step 2: CC Conflict Minimization We start with a single segment corresponding to the entire video. Then, for each pair of conflicting CCs (u, v) in O_{conf} that co-exist in the current segment, we identify an interval of time such that if we split the video at any frame within that interval, u and v will go to different partitions thus resolving the conflict. The next step is to count the number of conflicts that can be resolved by splitting the video at each frame in the current segment. We greedily choose the frame where the maximum number of conflicts will be resolved as the next split point, and then we apply this procedure recursively on each partition until a stop condition is achieved. We do not split a segment if it contains less than $p^{\perp}_{conf} = 3$ conflicts or if splitting the current segment would create a segments shorter than $p^{\perp}_{time} = 25$ seconds.

We observed that this procedure tends to select intervals closely related to erasing events. More specifically, they are typically chosen at points where the speaker starts writing again after erasing content from the whiteboard.

Video Summarization. Given the set of video segments P^R found by conflict minimization, we use our spatio-temporal index to generate one key-frame for each segment to form the set of summary frames F^S. Instead of simply selecting frames from the segment, we compute each summary key-frame f^s_i by rendering the images of all CC groups that existed within that video segment. We use the timeline of each CC group to identify and render the latest version of its image during that video segment. In cases where the video segment still contains conflicting CC groups, we only render those that existed on the whiteboard closer to the end of the segment and have no conflicts among themselves.

4.2 Experiments

In our experiments, we compensate for the lack of standard datasets for this application by creating our own labeled dataset from an existing collection of linear algebra videos. These videos were recorded using a still camera with a resolution of 1920x1080, and the whiteboard always represents the largest element in the image. There is no zooming or panning. A total of 12 videos were manually labeled by 4 graduate students. Labeling each video required between 12 to 15 hours of work to define: ideal segments, best key-frames per segment, background objects per key-frame, unique portions of content that repeat through multiple key-frames and the pixel level ideal binarization for each handwritten content region. A total of 5 videos were used for training and the remaining 7 were reserved for the final evaluation. We used the training videos to adjust each of the parameters of our proposed pipeline, and whenever it was possible we used automated procedures to learn the values that would maximize our evaluation metrics on the training set. The newly labeled videos along with the ground truthing tools will be made publicly available.

To evaluate the effectiveness of the proposed method at different stages, we use 4 baselines: binary, whiteboard segmentation, ground-truth based whiteboard segmentation, and reconstructed. For each of these baselines, we uniformly sample 1 frame every 10 seconds of video. In addition, a fifth baseline, Max Sum, is added to compare an alternative strategy for key-frame selection after frame reconstruction using a sliding window to find key-frames with the maximum number of ink pixels within a 25 seconds window [20]. For each summary, we match CCs between ground truth and summary key-frames with overlapping time intervals. For every pair of frames, we find a translation of the summary key-frame that maximizes pixel-wise recall, and then evaluate pixel-wise recall and precision of overlapping CCs. We accept inexact matches between groups of CCs if their combined images match with pixel recall and precision above 50%. We chose this threshold to compensate for variations in thickness for good readable matches. For global recall metrics, we computed our metrics based on unique CCs in the ideal summaries, while global precision is measured in terms of all CCs in summary frames.

In Table 4.1 we present average results for different summarization methods. We consider number of frames, and the global and average per-frame recall/precision. As expected, the binary frames obtained the highest recall at 98.96% with the lowest precision because of all the non-content CCs.

Whiteboard Segmentation increases global precision from 64.01% to 70.32% with a small drop of 0.03% in global recall. Ground-truth based whiteboard segmentation suggest it would be possible to achieve precision of 73.27% with an ideal whiteboard segmentation method. Then, after analyzing stability and removing unstable CCs that belong to the speaker, the reconstructed binaries achieve a global precision of 94.28% which represents almost 24% increase in precision with a drop in recall of just 2%.

Our proposed method using conflict minimization obtains a better compression rate (50% fewer

Table 4.1: Results for different summarization methods in 7 test videos.

	AVG	AVG Global		AVG Per Frame	
Method	Frames	Rec.	Prec.	Rec.	Prec.
Initial Binary	295.71	**98.96**	64.01	**98.69**	63.30
Whiteboard Segmentation	295.71	98.93	70.32	98.43	69.87
Ground Truth Whiteboard Segmentation	295.71	98.94	73.27	98.49	73.29
Reconstructed Binary	295.71	96.95	94.28	96.49	90.51
Minimization of Conflicts	**17.29**	96.28	93.56	95.73	**92.21**
Maximum Sums [20]	34.42	96.49	**94.51**	96.13	91.95

key-frames on average) than selecting key-frames using the max sum method [20]. It also keeps global recall and precision almost at the same level. One of the reasons why our generated frames get slightly weaker global precision is because they render all non-conflicting CCs that exist on a given interval of time. This means that all noisy CCs that might exist on a given segment will be included in the corresponding summary frame. The same does not happen with sampled key-frames.

One issue with the local sliding window is that sometimes it can generate redundant key-frames if the window is too small. Our proposed method avoids this issue by optimizing key-frames globally instead of locally. Our approach can further compress the whiteboard content by placing non-overlapping regions of contents on a single frame. However, recall can drop if the video is under-segmented and conflicts still exist in a segment because our method will only display the most recent elements.

4.3 Summary

We have proposed a novel method for lecture video summarization based on minimization of conflicts between regions of content. Our proposed CC stability analysis for reconstruction of binary frames and background removal procedures are very effective in increasing the precision of content extracted from lecture videos after binarization. In Chapter 6, we use the key-frames produced by this method for indexing and retrieved of math lecture videos.

As future work, we would like to test the effectiveness of this method using blackboard/chalkboard videos. We would also like to further extend the proposed method to work on harder cases where there is camera zooming/panning and lectures recorded using multiple cameras.

Chapter 5

Mathematical Symbol Recognition

In pattern recognition there are several applications for automatic recognition of math expressions. This is a hard task specially if the input data is handwritten. This task has two general modes: on-line and off-line. The first one refers to the case when tracing information is available. The second one refers to the case when tracing information is not available but images of the final drawings are provided instead. In Chapter 6, we require recognition of handwritten math symbols from images of typeset LaTeXexpressions and handwritten whiteboard content.

In this Chapter we are focused on using adaptations of off-line features for on-line classification of math symbols [25]. Since the training dataset is usually biased toward very few classes, we also reduce this class-representation disparity by generating synthetic data using an elastic distortion model. We tested our approach using different classification methods: AdaBoost.M1 [103] algorithm using C4.5 decision trees [96] as the weak learner, Random Forests [12] and Support-Vector Machines (SVM) [22, 128] with linear and Radial-Basis Function (RBF) kernels. The input of these classifiers is a set of simple features used for description of the shapes of each symbol. For our main experiments we used datasets from the Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME) 2012 [75] and CROHME 2013 [78]. Finally, in order to provide a comparison of our approach with other existing approaches for isolated math symbol recognition, we benchmarked our system using the MathBrush database [68]. The complete source code of our system is available at `http://www.cs.rit.edu/~dprl/Software.html`.

Different approaches have been used before in the field of math recognition and retrieval as documented by Zanibbi and Blostein [137]. The CROHME competitions [74, 75, 78] provide a good comparison between different approaches for math expression recognition on a set of standardized tasks. For the scope of this paper, we are only interested in the specific problem of isolated on-line math symbol recognition.

One of the most important elements of a math symbol classifier is the set of features used to describe each symbol. In the survey by Yang et al. [130] different feature extraction techniques for shape description have been identified. While some feature extraction techniques based on timing or drawing information have been used before for on-line symbol recognition [5, 6, 34, 39, 118], techniques based on shape description might be more robust since a single shape can be drawn in several ways. In the work by Delaye et al. [28] a set of 49 features that describe both the drawing process and the shape have been proposed as a baseline for different on-line symbol recognition datasets.

In addition to strong features, good training data is also important. For some applications, synthetic training samples can be generated to improve the quality of a given training set. In the current application, it is possible to apply a data generation model to create synthetic samples from the existent. In the work by Simard et al. [108], an elastic distortion model for data generation is applied to off-line character recognition. Also, the Sigma-lognormal model of the kinematic theory of rapid human movement presented in the work by Plamondon et al. [94] has been used to produce very realistic synthetic on-line signatures and handwritten gestures.

The quality of a training set is not only determined by its size, but also by having a good representation for each symbol class. In this sense, a balanced representation of classes is helpful to make symbol classifiers that base their decisions mainly on the actual shapes of the symbols rather than based only on the prior probabilities for each class. We would like our sample to be representative and to have a balance between classes. Ideally, a good data generation strategy can enhance a dataset even with limited amounts of synthetic samples.

Nevertheless, it has to be considered that on a recognition problem there might be different styles for a single class [101]. For handwritten symbols, these styles go beyond small differences in slant and thickness to the point of having completely different shapes that still belong to the same class like for example the lowercase z character written in cursive style. If a given writing style is absent from the training set, then it will be very difficult or even impossible for a classifier to learn to correctly identify symbols using that style.

In our work we use a data generation strategy based on balancing class representation. The CROHME [75, 78] training and testing datasets are highly biased. For this reason, we used average per-class accuracy as an indicator of how the system performs on the individual classes.

5.1 Strategy for Data Generation

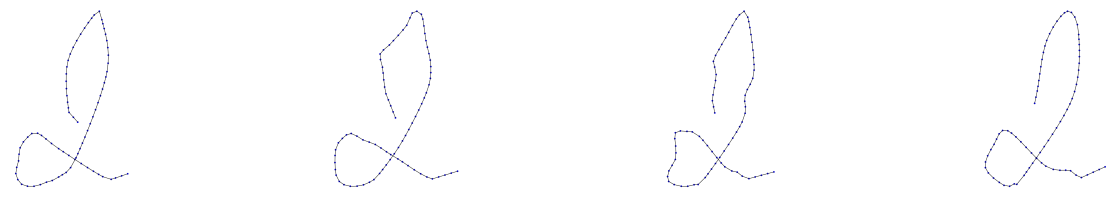
Given a distortion model and a training set, additional data can be generated in any convenient way. Ideally, more data can produce more accurate classifiers. However, there is no guarantee that higher accuracy will be achieved by getting more samples from what is already in the training set. In fact, we could easily obtain lower accuracies by multiplying noisy samples. In this work we use a data generation strategy that simply balances the class representation in the training set.

Our strategy for data generation is very simple: all classes should have at least a given number of samples. The main parameter for this method is the percentage T of the size of the largest class C that will be the new minimum of samples per class $min\_count = T * |C|$. For each class, if the number of elements is smaller than min_count, then the existing samples are used to create distorted copies of them until the size of the class is equal to min_count. Note that if $T \geq 1.0$ then the resulting dataset will be perfectly balanced.

For the generation of synthetic samples we use an elastic distortion model similar to what Simard et al. [108] used before for symbol recognition. One important difference in the current model is that our input data is on-line instead of images. In this sense, a distortion can be easily created by just randomly moving the points of each trace around their original positions. However, if the points are moved too far away from their original positions, or if two contiguous points are moved in completely opposite directions, the resulting sample might not be a realistic variation of the original.

In order to attempt to create realistic variations of the original shapes in our system, we used Perlin Noise [90] to generate smooth noise maps that control how much should we move each point in the shape, and because these noise maps are smooth, contiguous points are very likely to move on similar directions making the final result credible. Besides the parameters of the Perlin Noise maps [90], the system defines the maximum level of distortion to apply to a given point.

(a) Original (b) Copy 1 (c) Copy 2 (d) Copy 3



(e) Original (f) Copy 1 (g) Copy 2 (h) Copy 3

Figure 5.1: Examples of results for our distortion model. (a) and (e) are the original symbols, and (b)-(d), (f)-(h) are distorted versions of each respectively.

This is the maximum distance that a point can travel from its original location, and it is defined as a proportion of the length of the diagonal of the bounding box that contains the trace being distorted. Empirically, we found that 30% of this length produces credible but highly distorted samples and that about 9% of this length was more than enough distortion to create visually acceptable variations with a positive impact in the training process. An example of our model applied to distort two symbols is shown in Figure 5.1.

5.2 Symbol Recognizers

Three factors must be considered for the creation of a recognizer of symbols for on-line math expressions. The first one is a preprocessing procedure to mitigate the noise present in the input data. The second factor is the set of features used to describe the symbols. Finally, the third factor is the machine learning technique used for classification and in our case we are comparing four: AdaBoost.M1 [103] with C4.5, Random Forests [12], and SVM with linear and RBF kernels.

5.2.1 Preprocessing of Input

Our preprocessing procedure is an adaptation of the method in [42]. Three main stages can be identified on the process: Removal of duplicated points and hooks, then smoothing of the trace and finally size normalization of the symbol. However, there are two minors differences in our approach. First, during the removal of duplicated points, we only remove them if the length of the sub-trace connecting them is smaller than 10% of the length of the diagonal of the bounding box that contains the whole trace. The second difference is the cubic spline used for the final re-sampling of the shape which in our case we use Catmull-Rom splines [15]. After size normalization, the resulting symbols will have all their coordinates contained inside a 2-by-2 box centered at the origin.

5.2.2 Features

One of our goals was to use features that were both simple and strong for shape description. Our input data is on-line which implies that tracing information is available and can be used to compute features that cannot be easily computed for off-line data. However, we avoided relying on information that describes how the trace was drawn because of the high degree of variation introduced by the different writing styles.

We considered different features and tested many combinations of them and finally we produced a set that is both simple to compute and achieves high accuracy in our training and testing sets. These features can be divided into four main categories: global features of the symbols (11), crossings features (30) 2D fuzzy histograms of points (25) and fuzzy histograms of orientations (36). A total of 102 values are used to describe every symbol in our system.

Global features of the symbols

These features represent individual values that describe the complete symbols and provide general information about their shapes. The current set of global features is: number of traces (1), angular change (2), line length (2), number of sharp points [42] (2), aspect ratio (1), mean x coordinate (1), mean y coordinate (1) and covariance of x and y coordinates (1). For angular change, line length and number of sharp points, two values are added: the total sum and the average of the values for all traces.

Crossings features

This kind of features has been previously used for off-line character recognition [119]. Given any arbitrary line, it is possible to compute the number of times that it intersects the traces of a symbol. Also, if the line intersects multiple times the traces, then the first and the last intersections can be useful to describe the shape at that location. If we extend this idea to a set of lines at fixed vertical and horizontal positions, then we get a set of features useful for classification of shapes of symbols.

However, one of the issues of these features is their sensitivity to small variations in writing style. A way to mitigate this issue is using the average of many parallel lines to describe a region instead of a single line. Currently, X and Y axis are divided into five regions each, and for each region nine sub-crossings are computed and the average of those sub-crossings is used to describe the region. Three average values are computed per region: count of intersections and positions of first and last intersections. The entire shape is describe horizontally and vertically using $5 \times 3 + 5 \times 3 = 30$ values.

Figure 5.2.a shows an example of horizontal crossings for the number 3. In this example, the lines in the middle represent the horizontal sub-crossings of that region of the symbol. Most sub-crossings on this region intersect the trace exactly once, but the last intersects three times. The example shows how little variations could lead to large changes for crossings values if only one line is used per region. However, by using the average of nine lines the final value of crossing counts for this example will be very close to one.

2D Fuzzy histogram of points

The normalized region that contains the symbol is divided into a grid of 4-by-4 cells, and then each point on each trace is assigned a fuzzy membership value for each of the four corners of the cell of the grid where it is contained. A grid of n-by-n cells will contain a total of $(n + 1) \times (n + 1)$

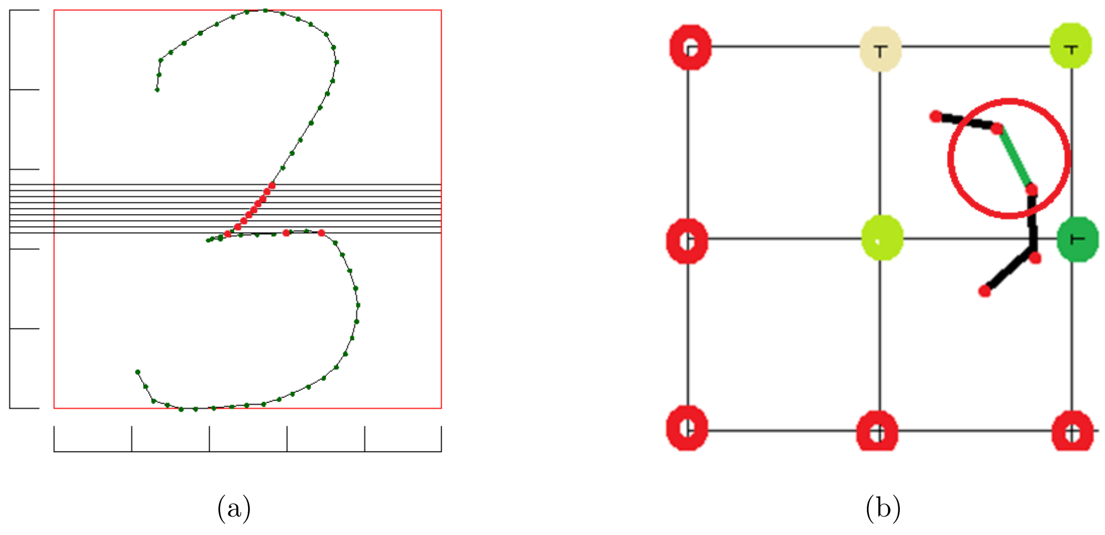


Figure 5.2: Two kind of features used by our system. (a) Horizontal crossings with average of sub-crossings. Each horizontal line in the middle represents a sub-crossing. Intersections are marked with larger dots. (b) Fuzzy histogram of orientations. The circled line segment is affecting the four corners of the cell where it is located and most heavily the one in the middle-right.

different corners. In this case, $5 \times 5 = 25$. For a given point $P = (x_p, y_p)$ and a given corner $C = (x_c, y_c)$ the membership value m_p is defined as $m_p = \frac{w-|x_p-x_c|}{w} \times \frac{h-|y_p-y_c|}{h}$, where w and h refer to the width and height of the cell respectively. At the end, all membership values are added on each corner, and then these values are divided by the total of points in the symbol producing a normalized 2D fuzzy histogram of the coordinates of the points of the symbol.

Fuzzy histograms of orientations

Histograms of orientations have been used before as a zoning approach for description of shapes for off-line optical character recognition [119]. The entire symbol area is divided into small regions using a grid of 2-by-2 cells. Similar to the previous feature set, the grid defines $3 \times 3 = 9$ corners, and for each of them the slopes of the closest line segments are grouped into four bins. The grouping criterion used is the angle formed by that slope and the horizontal axis. Note that any slope can be represented by an orientation angle in the range $\left[-\frac{\pi}{8}, \frac{7\pi}{8}\right]$, and then that range can be divided into four bins of size $\frac{\pi}{4}$. In that sense, we could think of the first bin $\left[-\frac{\pi}{8}, \frac{\pi}{8}\right]$ as the representation of horizontal lines in the region. Similarly, the third bin $\left[\frac{3\pi}{8}, \frac{5\pi}{8}\right]$ represents the vertical lines, and the other two bins represent the two diagonal directions. These features consists of a total of $9 \times 4 = 36$ values.

Using a grid as described before, we create fuzzy histogram of orientations of line segments by using different weighting criteria. Consider that the middle point of each line segment falls inside of a cell in that grid, then the first weighting criterion determines the membership of the line segment for each of the four corners of the cell based on the relative distance to each corner. This membership system is illustrated in Figure 5.2.b. and is computed with the same formulas used for m_p in the 2D fuzzy histograms of points by defining the point $P = (x_p, y_p)$ as the middle point of each line segment. The second weighting criterion is the actual slope angle and it makes each line segment a member of the two angular bins with the closest ranges to its orientation. The distance between a given angle and a range is computed using the absolute angular distance between that angle and the value in the middle of that range. Note that angular distance is circular and some slopes can become members of both the first and the last bins. In summary, a single segment will affect a total of 8 bins, 2 angular bins per each of the 4 corners of the cell where its middle point is located.

The third weighting criterion is the relative length of the line segment. Initially, individual histograms are computed for each trace of the symbol and the contribution of each line segment is given by its own length divided by the total length of that trace. The next step is to combine the individual histograms of the traces into a single histogram for the whole symbol, and this is done by weighting the entire histogram of each trace using the length of that trace divided by the sum of the lengths of all traces. After re-weighting the individual histograms, these can be added up into a single histogram.

The fuzzy histograms are expected to be more tolerant to small variations that their non-fuzzy versions. Consider the case of a value that lies in the border between two bins in a non-fuzzy histogram. A very small change in the value could make it fall into a different bin. However, for the fuzzy case such change will only represent a small modification to the contribution of that value to the closest bins.

5.2.3 Classifiers

Four alternative classification algorithms were considered. Each of them has its own advantages and disadvantages. For example, the most accurate requires more time to classify a symbol, and the ones that provide faster classification times can also require longer training times. The following subsections define each of them.

AdaBoost.M1 with C4.5 Decision Trees

AdaBoost requires the use of a weak learner with at least 50% of accuracy and C4.5 decision trees fulfill this requirement in our current datasets. Of all the different versions of AdaBoost, we are using the AdaBoost.M1 algorithm [103]. The greedy approach described by Quinlan [96] for training of the individual decision trees based on the calculation of the gain ratio has been adapted to work with weighted samples. Also, when the tree is evaluated, the class probabilities are no longer computed using the counts of samples per class, but instead using the total weights per class divided by the total weight in the current node for use with boosting. Each decision tree is pruned using the pessimist error estimation with confidence intervals for the binomial distribution as done by Quinlan.

The main idea of boosting is that higher accuracy can be achieved by combining the output of multiple weighted classifiers. The algorithm is based on rounds were each round attempts to classify correctly the samples that were misclassified on previous rounds using a weighting system for each individual sample. Additional details can be found in work by Schapire [103]. In our tests, we set the number of boosting rounds to a maximum of 50.

Random Forests

A different ensemble method based on decision trees. It has the advantage of high parallelization as each decision tree on the ensemble can be trained and evaluated independently of the others.

We used the implementation of Random Forests included in the scikit-learn [89] library for python. It trains each tree using a bootstrap sample drawn from the training set, and randomizes the available attributes at the time of computing the best splits. No pruning is performed over the learned decision trees but their depth is limited. For our classifier, we used 50 trees with depths above 20. We set the limit of available features at each split to 30.

Support Vector Machines

An adaptation of SVM for multi-class classification [128] is being used in our system. We use the implementation provide by the scikit-learn [89] library for python. Two different kernels were tested: Linear and RBF. SVM with linear kernels can be easier to configure than SVM with RBF kernels. In the other hand, SVM using RBF kernels usually achieved the highest accuracy once that the right values for their C and Gamma parameters were found. The C parameter defines the simplicity of the model while Gamma controls the influence of each training sample. In our work we applied grid search to find good values for these parameters which achieve the highest accuracy for each training set. The greatest advantage of SVMs is that their accuracies are usually the highest and can be used as a reference when optimizing the parameters of other classifiers. The greatest disadvantage of SVM is their evaluation time for any given sample which might be due to the fact that several SVM trained for binary classification must be evaluated before deciding the final class of the sample.

5.3 Adaptations for Off-line Math Symbol Recognition

The method discussed in the previous section has been designed for on-line math symbol recognition. However, it is possible to adapt the preprocessing step in order to use it for off-line math symbol recognition. First, we render the on-line training samples as images. Then, we extract the contours from the off-line image and use them as the new symbol traces, and we train the system to recognize symbols based on these contour-traces.

To classify a new off-line handwritten math symbol, we simply extract the contours from a padded image of the symbol, and we apply our symbol recognizer trained using contour-traces. Note that our method works well with contour traces because most of our features are shape-based and do not depend on the temporal information of the original traces.

Similarly, we can use our approach to recognize off-line typeset math symbols by generating typeset training samples for each desired class using LaTeX. In this case, multiple fonts per class need to be considered during the rendering process. Also, special symbol classes like brackets, parentheses, and roots need to be generated for different sizes of contained elements. In addition, the data generation process discussed in Section 5.1 can be applied to generate of the contour traces, or simpler scaling and rotation distortions can be applied to the images themselves before extracting the contours used for training.

5.4 Experimental Results

For the first set of experiments with synthetic data generation we used the training set from CROHME 2013 [78] that contains 68598 valid samples of handwritten symbols, and two testing sets, one from part III of CROHME 2012 [75] with 6336 samples and the second is from CROHME 2013 with 6082 samples. The main different between these testing sets is the total number of classes.

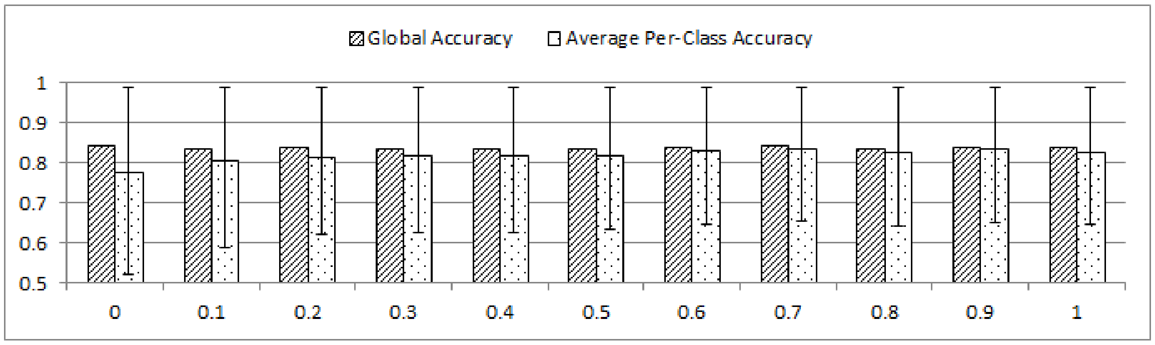


Figure 5.3: Effect of generating different amounts of data. The horizontal axis represent the percentage (T) of the size of largest class in the original training set. Note that for CROHME 2013 (101 classes) the global accuracy does not improve. However, the average per-class accuracy increases from 77.79% when $T = 0.0$ to 83.30% when $T = 0.7$. Also, its standard deviation decreases from 25.49% to 18.01%. No further improvements were obtained with larger values for T.

For CROHME 2012, 75 different class of math symbols are included. In the case of CROHME 2013, 101 classes are available. It is important to mention that these 101 classes are a super set of the original 75 in the first testing set. A custom training set was obtained by extracting the samples of the 75 common classes from the CROHME 2013 training set and the classifiers trained with this data were evaluated using CROHME 2012 testing data. This second training set has a total of 65544 samples.

Each of the original CROHME training sets is heavily biased toward very few classes. For example, only 5 classes (-, 1, 2, +, x) out of the 101 present in the CROHME 2013 dataset represent more than 35% of the whole dataset and just by adding 4 more symbols (=, "(", ")", 3) they account for more than half of the entire dataset. A similar bias is also present on the testing data. In order to produce less biased classifiers and at the same time attempt to reach higher recognition rates we applied our data generation strategy to balance the class representation. We tested different amounts of synthetic data as shown in Figure 5.3. As we expected, additional samples stop contributing to the accuracy of the system after a large number of samples has been created. For each test we include the global testing accuracy and also the average per-class accuracy with its corresponding standard deviation. From these results we decided that we would use a maximum of 70% of the largest class as the new minimum size per class for balancing on the CROHME 2013 (101) training set, and a maximum of 60% for the CROHME 2013 (75).

Given that the expanded version of each training set is relatively large compared to the original, 68598 vs 451637 for CROHME 2013 (101) and 65544 vs 291192 for CROHME 2013 (75), and that for each symbol a total of 102 features are used, we considered the use of Principal Component Analysis (PCA) for dimensionality reduction. However, this led to smaller accuracies even after using up to 99% of variance. For this reason we decided to keep the original set of features for our current analysis. Figure 5.4 shows a comparison of the global accuracy of the four classifiers over two different testing sets. It also shows that global accuracy values were usually lower when the extended training sets were used. SVM classifiers achieved the highest accuracies, specially the one using RBF kernel. Random Forest usually achieved higher accuracy than AdaBoost with C4.5. The best testing accuracies are 85.89% for CROHME 2013 and 94.49% for CROHME 2012. However, a direct comparison with CROHME results might not be possible as CROHME 2013 [78] does not provide accuracy for isolated symbol classification. In the case of CROHME 2012, the best

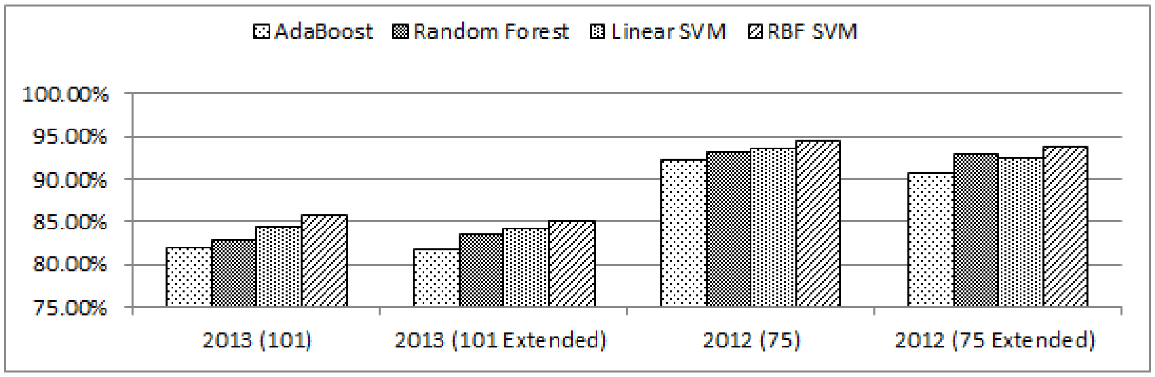


Figure 5.4: Comparison of global accuracy using different classifiers. Here, 2012 and 2013 refer to the testing set used while (75), (101) and extended refer to training set and whether that training set was expanded or not.

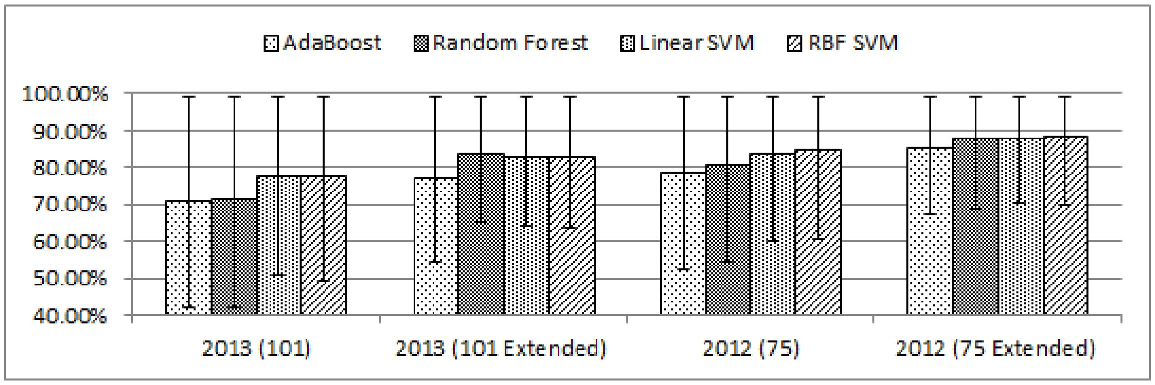


Figure 5.5: Comparison of average per-class accuracies using different classifiers. The extended training set always achieve higher average per-class accuracy

accuracy reported on the original competition was 96.85%, but this value is calculated over a subset of correctly segmented symbols from entire math expressions and not from isolated symbols. Figure 5.5 compares the different average per-class accuracies and their standard deviation. Extended training set always produced higher average values with smaller standard deviations.

A very important concept to keep in mind when balancing classes is that some symbols are ambiguous by nature. For example, s and S are symbols that can only be distinguished using context information. By plotting the accuracy of individual classes before and after the expansion of the dataset we can observe how certain classes increase in accuracy while others decrease. Figure 5.6 shows the plot for CROHME 2013 (101) including only classes with more than 5% of difference in accuracy. Current results suggest that ambiguous classes make trade-offs during the balancing process.

Considering the existence of ambiguous classes, we decided to recompute the global accuracy of our best SVM RBF classifiers if errors between ambiguous classes were ignored assuming that these could be solved later using context information. The ambiguous pairs of classes were the following: x-X, x-\times, X-\times, 1-|, (-|,)-|, 1-(, 1-), 1-/, 1-COMMA, c-C,)-COMMA, p-P, \prime-COMMA, \prime-|, v-V, s-S, q-9, o-0, \prime-/, /-COMMA. We achieved 93.52% for CROHME 2013 (101) and 96.36% for CROHME 2012 part III.

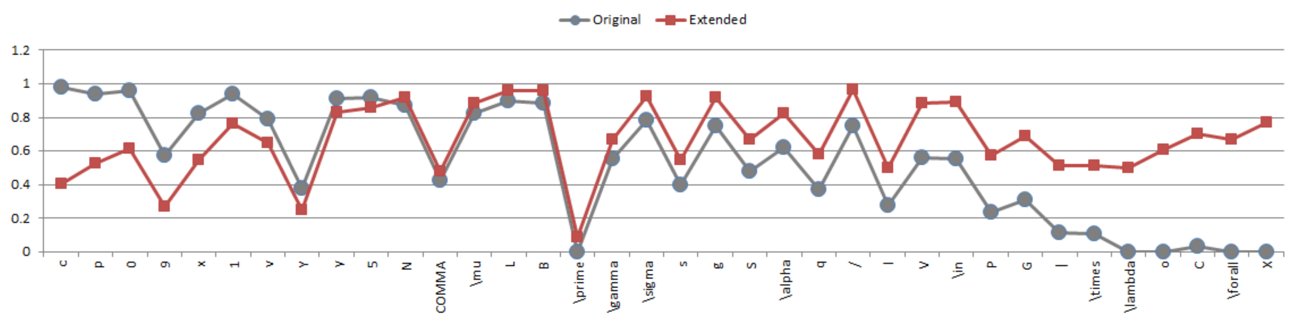


Figure 5.6: Classes with more than 5% of change in accuracy due to extension of the training set. Note that classes like "x" suffer a great loss of accuracy while classes like "X" and "times" have a great gain.

Table 5.1: Comparison of accuracy of different methods on MathBrush

Method	**Classifier**	**Top - 1**	**Top - 5**
Hu et al. [39]	HMM	82.9%	97.8%
Alvaro et al. [5]	RNN	89.4%	**99.3%**
MacLean et al. [67]	Greedy DTW	85.8%	99.1%
Proposed Method	AdaBoost C4.5	88.4%	98.7%
	Random Forests	87.9%	98.4%
	SVM Linear Kernel	88.6%	99.1%
	SVM RBF Kernel	**89.8%**	99.1%

Since direct comparison with results from other systems using CROHME datasets is difficult, we tested our system using the MathBrush [68] database. The database contains 100 classes of symbols, but following the procedure of other approaches using this dataset [5, 39], seven classes of symbols were discarded: $\leq, \neq, <, \lambda, \Omega$, comma and dot. Table 5.1 contains a comparison of Top-1 and Top-5 accuracies for the following approaches on MathBrush dataset: Hu et al. [39] using Hidden Markov Models (HMM) and Pen-Up/Down features, Alvaro et al. [5] using Recurrent Neural Networks (RNN) and hybrid features, MacLean et al. [67] using greedy Dynamic Time Warping (DTW), and finally our approach using the four classifiers described in section 5.2.3. We tested each classifier using 20 experiments as described in [5] using exactly the same partitions of data. Note that in the work by MacLean et al. [67] only 70 classes of single-trace symbols were used.

5.5 Summary

We have presented an approach for on-line recognition of handwritten math symbols. We compared the accuracy of our approach using different machine learning techniques and obtained relatively small differences in terms of global accuracy. Math symbols are described in our approach mainly using off-line features on on-line representations. It is worth to mention that our initial tests suggested that crossings features are by far the strongest in our current set of features. Our features mainly aim to describe the shapes of the symbols, and we achieve recognition rates that are competitive with other approaches that use the on-line data to compute time-based features.

Our experiments shown that for certain datasets like CROHME, data generation might produce subtle benefits that might be missed if we only pay attention to the global accuracy of the system. In most of our test, extended datasets created classifiers with slightly lower global accuracy but better average per-class accuracy with smaller standard deviation. This loss of global accuracy was mainly due to the existence of ambiguous classes in the dataset as trade-offs took place between pairs of ambiguous classes.

The adaptations for recognition of off-line math symbols described in Section 5.3 are used in Chapter 6 for recognition of symbols from images of typeset LaTeXimages and handwritten whiteboard content.

Chapter 6

Retrieval of Mathematical Notation in Lecture Videos

In this chapter we present the Tangent-V (Tangent-Visual) model for retrieval of mathematical notation found in lectures videos. As discussed in previous chapters, there are many cases where a person might want to search for relevant content in videos using a mathematical expression as the query. For example, a linear algebra student wants to find all portions of lecture videos where the identity matrix is used, and might start a search using I_n as the query. Our method is capable of finding relevant matches for such queries using an image-based search model that works for images of handwritten (whiteboard content) and typeset (rendered $\LaTeX$) mathematical expressions

As discussed earlier in Chapter 2, some methods exist that summarize whiteboard content extracted from videos by using OCR techniques. However, OCR is limited when it comes to mixtures of mathematical expressions and text because of the lack of explicit segmentation between these two kinds of content, and the increased number of potential symbols that we find in math. Also, full recognition including parsing of handwritten math expressions is still far from being a solved problem [79].

We propose a method that does not rely on full recognition of mathematical expressions from images. Instead, we use Line-of-Sight graphs to create a rough representation of the structure of whiteboard content, classify math symbols using a probabilistic math symbol classifier (see Chapter 5), and index the expressions using multiple combinations of potential labels for pairs of elements from the graphs. This method allows retrieving expressions even if the real classes for some symbols are not matched with very high confidence as long as there are enough symbols that can be matched between the query and a given candidate key-frame.

Tangent-V is a generalization of the Tangent-S symbolic formula retrieval model discussed in Chapter 3. Like the original, the system indexes formulas based on pairs of elements. It also represents each expression using a graph-based representation, and the model uses two layers that work very similarly to the original model: initial candidate selection from the index, and detailed matching with re-scoring and re-ranking.

The first main difference is that Tangent-S depends on a hierarchical tree structure which is provided by the source Presentation or Content MathML found in a document. On the other hand, that hierarchy is not given for lecture video key-frames and Tangent-V uses a more general Line-of-Sight graph to avoid relying on segmentation and parsing methods that would fully recognize the expression but have low success rates. Second, for each pair of symbols in Tangent-V, the model does not know before hand the real label of each symbol, and for this reason it uses multiple combinations of labels generating multiple index entries for a single pair of symbols. Third, there are not known edge labels in the Line-of-Sight graph, and Tangent-V replaces the relative path labels with a 3D unit vector system that represents the local relationship between two nodes. Finally, the retrieval targets in documents are well delimited formulas, but no formula regions are given in the case of lecture videos, meaning that Tangent-V has to work with non-delimited retrieval targets. In Tangent-V, an entire key-frame is the smallest retrieval unit which might contain full and/or partial matches of the target formula at multiple locations.

The proposed method has some limitations. First of all, Tangent-V is very effective in finding targets in cases where the symbol class confidences for a shape in the query are very consistent with the symbol class confidences for the same shape in the index. However, performance degrades when variations in handwriting style cause the classifier to provide different symbol class confidences for a distorted version of the same symbol. Also, the lower the confidences for a given shape, the more combinations of pairs of symbol labels that are tested for matching making the retrieval of distorted shapes much slower than in cases where the symbol classes are correctly identified with high confidence.

In this chapter we explore using the Tangent-V model for retrieval of handwritten mathematical expressions from video key-frames using images of handwritten expressions as queries (**HW-to-HW**). We also consider a variation where LaTeX expressions extracted from the corresponding class notes for each lecture are rendered as images and are indexed along with the original symbolic representation using the Tangent-V model. In this case we can use symbolic LaTeX expressions as queries by rendering them to images first and then using the same retrieval model that we would use for handwritten images (**LaTeX-to-LaTeX**). Then, cross-modal search is also considered by allowing LaTeX expressions to be used as queries to find handwritten content in key-frames (**LaTeX-to-HW**). Finally, we also allow a cross-modal search where images of handwritten expressions can be used to find symbol LaTeXusing the index of rendered LaTeX expressions (**HW-to-LaTeX**).

6.1 Methodology

Lecture videos containing mathematical notation need to be preprocessed in order to extract the handwritten whiteboard content as binary images. Then, we construct a graph based representation for the content, and using a symbol classification method, we index the content using pairs of symbol labels based on graph edges. Finally, the Tangent-V model uses two-layers for retrieval of math notation from images: the first layer does a fast candidate selection based on pairs of symbol labels extracted from the query. The second layer applies a detailed matching procedure that merges the initial candidates into larger unique matches.

6.1.1 Preprocessing

First, the raw lecture videos need to be summarized by extracting the handwritten content from the whiteboard as binary images. For this purpose, we use the lecture video summarization approach described in Chapter 4. This method creates a small set of binary key-frames for each lecture video. We use our on-line handwritten math symbol classifier with the adaptations discussed in Section 5.3 to recognize the math symbols found in these binary images using Random Forests for probabilistic symbol classification. The confidences for the top classes of each connected component are obtained using this method.

Some lecture videos might be accompanied with an external set of notes describing the lecture content as an auxiliary document. In these cases, the math expressions found in these documents can be extracted and indexed using Tangent-S (Chapter 3). We can also use the extracted expressions to find the set of unique math symbols classes that are expected to appear on the corresponding lecture video. Typically, these represent only a small subset of math symbol classes. For each lecture, we split the training data into two sets, one containing samples only from math symbol classes expected for that lecture, and the other portion containing samples for the remaining classes. We train a specialized math symbol classifier for that particular lecture using the data from the expected classes and adding the remaining data as one single rejection class labeled "Junk" used to reject unexpected symbol shapes. If no lecture notes are available, the system can still used a general math symbol classifier trained for all classes and without rejection class. In our current experiments, we only considered videos for which these lecture notes were available.

The LaTeX expressions extracted from the auxiliary notes are rendered as binary images for later indexing. The LaTeX adaptation of the math symbol classifier described in Section 5.3 is used to train specialized LaTeX math symbol classifiers per lecture. These are then used for recognition of the symbols on each of the binary images of the LaTeX expressions of the lecture.

6.1.2 Graph-based Representation

Representing images of math expression using SLTs or OPTs as we do for symbolic formula retrieval is hard because no concrete symbols or relationships are given in this case. Also, because the math expressions are not delimited, a hierarchical structure is hard to build from raw data. A full math expression recognition approach that considers segmentation and classification of regions as either text or math regions might be required as a preprocessing step in that case. We avoid these recognition requirements by using a more general graph-based representation for binary images based on Line Of Sight (LOS) graphs.

The LOS graphs have been used previously for on-line handwritten formula recognition [40]. In general, they tend to capture most relevant connections between pairs of elements in a visual representation. In this context, we define each connected component in the binary image as one vertex in the graph. Then, we build the edges of the graph by taking one vertex at a time as the observer, then we compute what other connected components can be seen (are in the line of sight) of this observer. We use the convex-hull of the contours of each connected component, then we define a direct line of sight between the center of the observer and each point in the convex-hull of each of the remaining connected components. We connect the observer node to any other vertex that can be seen by at least one of these lines between the eye and its convex hull points without being blocked by other vertices. Figure 6.1 contains examples of the LOS graphs for three different math expressions.

There are some drawbacks of using the LOS graph representation for binary images of whiteboard content. The first major drawback is the edge density. Since whiteboards are documents without a restricted format and content can be written at any portion of the board, a single connected component is very likely to be able to see multiple unrelated connected components written at distant locations of the whiteboard as shown in Figure 6.2(a). The second draw back is the number non-relevant elements drawn on the whiteboard that become part of the graph like underlines, and division lines that are meant to separate regions of content but are not part of the lecture content itself. Finally, single connected components that represent multiple (merged) symbols or multiple connected components that represent a single symbol (split).

To deal or mitigate some of the issues previously described, we proposed some post-processing steps that refine initial line of sight graph based. First, we compute the median width $\tilde{w}$ and the median height $\tilde{h}$ of all connected components in the graph, and we set a maximum distance threshold $E^{max} = 2.0 \times max(\tilde{w}, \tilde{h})$, that will remove connections between pairs of vertices that are at a distance greater than E^{max} from each other. This simple rule removes a large number of irrelevant

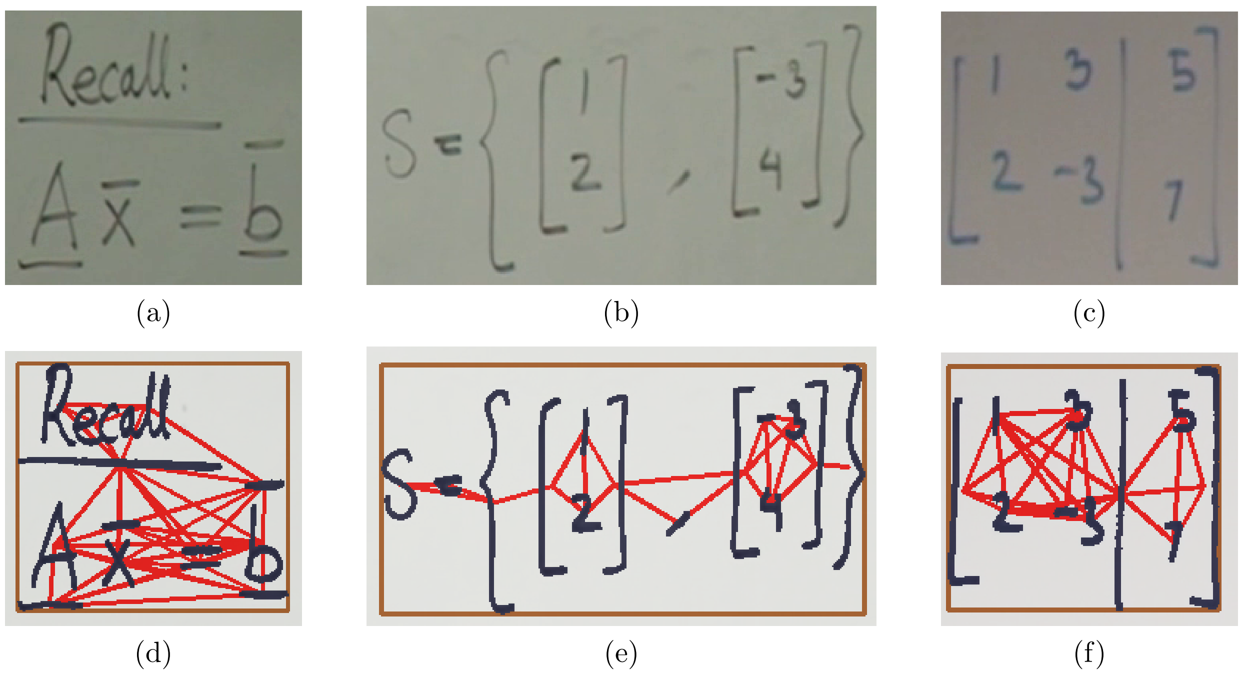


Figure 6.1: Line-of-Sight graph for image-based formula representation. Panels (a)-(c) are the original images, and (d)-(f) are their corresponding Line-of-Sight graph representation using connected components as nodes.

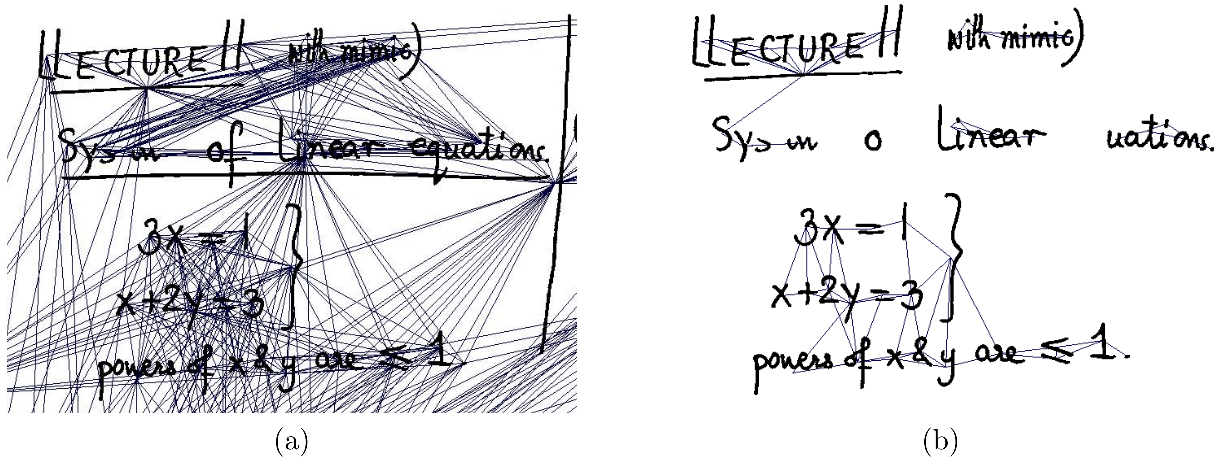


Figure 6.2: Raw Line-of-Sight graph and Filtered Line-of-Sight graph. These two images show the corresponding line of sight graph for the same region of a key-frame using (a) LOS graph without post-processing, and (b) LOS graph after post-processing.

edges from the initial LOS graphs. The second step is to remove large connected components that are most likely content separators from the graph and very small components that could be noise. For a given connected component v with width w and height h, we its horizontal relative scale $s_{\tilde{w}}$ as $s_{\tilde{w}} = \frac{w}{\tilde{w}}$ and its vertical relative scale $s_{\tilde{h}}$ as $s_{\tilde{h}} = \frac{h}{\tilde{h}}$. We establish two criteria based on size: The maximum of the two scale factors must be greater than the minimum scale, and neither scale factor (horizontal or vertical) can be greater than the maximum scale. This is:

$$\mathit{keep}\left(v\right) = s^{min} \leq \mathit{max}\left(s_{\tilde{w}}, s_{\tilde{h}}\right) \wedge s_{\tilde{w}} \leq s^{max} \wedge s_{\tilde{h}} \leq s^{max}$$

where $s^{min} = 0.1$ and $s^{max} = 10.0$ represent the minimum and maximum relative scale thresholds. Note that this size-based filter is only applied to binary images of lecture video key-frames, but it is not used for rendered LaTeX and for all query images in general. An example of a refined LOS graph can be seen in Figure 6.2(b).

To deal with many cases of splitted CCs, we generate an auxiliary K-NN graph were each CC is connected to its closets K neighbors. In this case, we use $K = 2$ to look up for potential merges between each CC and its closest neighbors. This K-NN graph is computed over the refined LOS graph. For each pair of elements u and v, a fake connected component x is created such that x combines the images of u and v into a larger image which no longer represent a real connected component of the binary image. if x meets the size criteria $\mathit{keep}\left(x\right)$, then the math symbol classifier is applied over the image of x. Let c_u^1, c_v^1, and c_x^1 be the confidences for the most confident labels for u, v and x respectively. The merged CC x will only be accepted if its most confident label l_x^1 is not the rejection class label "Junk" and the confidence for this label is greater than the average of the confidences of the top individual top labels for CCs u and v, $c_x^1 > \frac{c_u^1 + c_v^1}{2}$.

Our model currently does not handle merged CCs, but we observed that in practice these are a lot more common on text and very rare in our handwritten mathematical equations data. From this point, every time we refer to the LOS graph of a binary image, we will be referring to the refined LOS graph after all the post-processing steps have been applied.

6.1.3 Symbol Pairs Index Model

We use the edges of the LOS graph representation to define pairs of symbols similar to what is done in the Tangent-S model described in Chapter 3. The main difference here is, however, that we do not always have a single label per each node of graph. Also, we do not have explicit edge labels, and therefore we need to use a different model based on unit vectors to represent the relative relationship between pairs of symbols.

For a given pair of nodes u and v from a LOS graph, we define the 3D unit vector $\langle d_x, d_y, d_z \rangle$ as the vector that indicates the spatial relationship between u and v. Most relationships between elements on the binary images can be modeled using a 2D vector $\langle d_x, d_y \rangle$ that takes one of them as the reference point and indicates the direction of the location of the other in relation to this reference point. In other words:

$$\langle d_x, d_y \rangle = \mathit{center}\left(v\right) - \mathit{center}\left(u\right)$$

where the function $\mathit{center}(x)$ returns the location of the center of the bounding box of a given node x, and u is being used as the reference point.

However, the 2D vector becomes unstable when one of the nodes partially or completely contains the other node. This is usually the case for root symbols and its operands, and is also common in cases where elements are circled on the board. To deal with these especial cases, a third dimension d_z is added which represents the closeness between node centers.

We define a sphere around the center of each node in the graph. For a given node u, with width w_u and height h_u, we define its sphere radius r_u as:

$$r_u = \frac{min\,(w_u, h_u)}{2}$$

Then, for nodes u and v, we choose the one with maximum sphere radius $r_s = max\,(r_v, r_u)$ as the constant radius of the sphere, and we compute d_z as the vertical component that keeps that radius constant as follows:

$$d_z = \begin{cases} 0 & if \ |\langle d_x, d_y\rangle| \geq r_s \\ \sqrt{r_s^2 - |\langle d_x, d_y\rangle|^2} & otherwise \end{cases}$$

which means that the d_z is only used in cases where the symbol centers are at a distance smaller than r_s from each other. We finally make the 3D vector $\langle d_x, d_y, d_z\rangle$ by dividing it by its norm. Note that under this definition the d_z component will always be positive or 0.

After defining the spatial relationship between two nodes, the next step is to generate multiple index tuples that will represent the same pair based on combinations of confidences for the top classes for each symbol. For a given graph node u, we define its minimum set of top k class labels $L_u = \{l_u^1, l_u^2, ..., l_u^k\}$ with their corresponding confidences $C_u = \{c_u^1, c_u^2, ..., c_u^k\}$ such that

$$\sum_{i=1}^{k} c_u^i \geq c^{min} \vee k = l^{max}$$

where $c^{min} = 0.8$ is a threshold of minimum total confidence of the top k labels, and $l^{max} = 10$ defines the maximum number of labels to consider. In this case, k will be selected based on whichever of the two criteria that produces the smallest set of labels. After applying the criteria, if the set of labels contains the Junk class, then we remove it from the set along with its corresponding confidence score.

The final set of symbols pairs for a given pair of nodes u and v is given by:

$$SortedPair(l_u^a, l_v^b) = \begin{cases} \left(ID\,(u,v)\,, l_u^a, c_u^a, l_v^b, c_v^b, \langle d_x, d_y, d_z\rangle, 1\right) & if \quad l_u^a \leq l_v^b \\ \left(ID\,(u,v)\,, l_v^b, c_v^b, l_u^a, c_u^a, \langle -d_x, -d_y, d_z\rangle, -1\right) & otherwise \end{cases}$$

$$Pairs\,(u,v) = \{SortedPair\left(l_u^a, l_v^b\right) \mid \left(l_u^a, l_v^b\right) \in L_u \times L_v\}$$

where $ID\,(u,v)$ is a function that assigns a unique identifier to each pair of graph nodes, and *SortedPair* is a function that generates the final tuple that will be used for indexing of a given pair of symbol labels. The function *SortedPair* imposes a unique sorting where in all cases the smallest label of the pair will be used as the fixed reference point which means that it needs to invert the unit vector (except for the d_z component) to indicate that the labels were inverted. Also, note that a final value of 1 or -1 is added to the pair to record if the original labels were inverted. For example, for both expressions 2^x and $2x$, the 3D unit vector will point from the center of 2 towards the center of x. This is due to 2 being considered a smaller label x in their lexicographic order. For a given graph with N_n nodes and N_e graph edges, if we define the maximum number of labels per node to be l^{max}, then we can generate a maximum of $(l^{max})^2 \times N_e$ raw pairs for indexing. In practice, this number is greatly reduced in cases where the top labels have very high confidence.

In our final index structure, we keep references to all lectures, key-frames and graphs indexed. We build an inverted index of symbol pairs based on the raw pairs obtained for every connection on each graph on the index. The entries of this inverted index are the pairs of labels l_u and l_v of a given raw pair, and the entry defines a posting list containing references to all instances of that given pair of labels using tuples of the following form:

$$(pair_{ID}, c_u, c_v, \langle d_x, d_y, d_z \rangle, dir)$$

Where $pair_{ID}$ is a unique global identifier that references a particular pair of elements from a graph in the index, c_u and c_v are the confidences for labels l_u and l_v respectively, and the value $dir \in 1, -1$ indicates if the unit vector was inverted from its original values.

6.1.4 Retrieval

Tangent-V uses a two-layer retrieval model where the first layer quickly finds candidate match locations, and the second layer applies a detailed matching procedure that finds the most consistent match locations from the initial set of candidates.

Note that the same index structure can be used to index both binary key-frames extracted from videos or rendered binary images of LaTeXexpressions. Most steps of the retrieval process work exactly identical for either case except the last step where unique matching regions are grouped across contiguous key-frames for lecture videos, or they are grouped by unique LaTeX strings for binary images of LaTeXexpressions.

Layer 1: Candidate Selection

The first layer of the retrieval process takes as input the set of all raw pairs extracted from a query binary image, and produces a set of aligned index matches from binary images stored on the index. This process is executed in 3 steps: index entry matching, index entry merging and match generation.

Step 1: Symbol pair matching. The first step uses the query symbol pairs to find all matching postings in the index. For each symbol pair in the query, we use the labels of its nodes l_u and l_v to find the corresponding entry in the inverted index. We then compute the cosine similarity between the unit vector D^q of the query raw pair and each unit vector D^i in the posting list for this entry. The cosine similarity is defined as the dot product of two unit vectors, and represents the cosine of the angle formed by the two vectors. We set a threshold of minimum cosine similarity θ^{min} between vectors to consider them as matching if and only if $D^q \cdot D^i \geq \theta^{min}$. In this case we set $\theta^{min} = \cos 45°$, meaning that the maximum angular difference between D^q and D^i currently tolerated is $\pm 45°$.

The final score for a given index pair $T^i = \left(pair^i_{ID}, c^i_u, c^i_v, D^i, dir^i\right)$ matching a query pair $T^q = \left(pair^q_{ID}, c^q_u, c^q_v, D^q, dir^q\right)$ is given by

$$Score\left(T^i, T^q\right) = \left(c^i_u\ c^q_u\right)\ \left(c^i_v\ c^q_v\right)\ \left(D^i \cdot D^q\right)$$

where $Score\left(T^i, T^q\right)$ produces a value between 0 and 1. Large confidences for labels on each pair and large agreement between unit vectors will result in matching pairs scored higher. We also compute the estimated alignment direction using $Direction\left(T^i, T^q\right) = dir^i\ dir^q$

Step 2: Aggregating Index matches by Pair IDs. Note that on step 1, we might find multiple matches between unique index pairs and unique query pairs based on all the consistent label assignments between their aligned nodes. Therefore, we need to aggregate these matches by unique combinations of $\left(pair^i_{ID}, pair^q_{ID}\right)$. The aggregation process computes the sum of the scores and directions for all the initial pairs.

Step 3: Constructing Initial Aligned Matches. This step will generate aligned matches for each aggregated pair found in the previous step. In this context, we formally define an **aligned match** as follows:

Definition (*Aligned Match*): We define an aligned match m_x as the triplet $m_x = (G^q_x, G^i_x, f_x)$, where G^q_x is a subset of nodes from the query graph G^q ($G^q_x \subseteq G^q$), G^i_x is a subset of nodes from a binary image graph G^i from the index ($G^i_x \subseteq G^i$), and f is a graph isomorphism mapping nodes from G^q_x to G^i_x.

The data structure used to represent an aligned match needs to be linked to the original references in the index, and the final direction score of the aggregated pair plays an important role in speeding up this process by indicating which element of the query pair should be aligned to which element of the index pair. For a given query pair defining the subset $G^q_x = \{u^q, v^q\}$ and a matching index pair defining the subset $G^i_x = \{u^i, v^i\}$, a positive direction value indicates that the mapping should be $f(u^q) = u^i \land f(v^q) = v^i$, while a negative direction value indicates that the mapping should be $f(u^q) = v^i \land f(v^q) = u^i$. Finally, the resulting matches are organized and divided by index binary image.

Layer 2: Formula Structure Alignment

The second layer of the retrieval process takes as input the initial aligned matches generated by the first layer, and produces a smaller set of larger aligned matches representing each matched formulas. Four steps are involved in this process of growing the final larger matches: greedy match growing, greedy match connection, redundant match filtering, and match grouping.

Step 1: Greedy Match Growing. In this step, the initial pair-based aligned matches are merged together to form larger matches based on a compatibility criterion. We say that two aligned index matches m_1 and m_2 are compatible for merging if they share identical mappings for at least one query node, and the remaining mappings from each match are not in conflict with each other. This is:

$$\mathit{Compatible}\left(m_1, m_2\right) = |G^q_1 \cap G^q_2| > 0 \land \forall(u, v) \in G^q_1 \times G^q_2 \left(f_1\left(u\right) = f_2\left(v\right) \iff u = v\right)$$

Any two aligned matches that do not have conflicting alignments (inconsistent mappings) can be merged using *Merge*:

$$\mathit{Merge}\left(m_1, m_2\right) = \left(G^q_1 \cup G^q_2, G^i_1 \cup G^i_2, f_1 \cup f_2\right)$$

The greedy matching proceeds as follows: for each binary image that contains at least one match found by the first layer, take the match with the current top score and start a greedy growing procedure. Find the next compatible match with the highest score and merge it to the largest match. Repeat the search until no more compatible matches are found. Remove the largest from the set of pending matches, and re-start the greedy procedure again with the next top score until all matches have been examined. The pseudo-code for this procedure can be found in Algorithm 1.

Algorithm 1 Layer 2: Step 1 - Greedy Match Growing

```
for all binary images containing a set of aligned matches $M^i$ do
    $M^f \leftarrow \emptyset$
    $M^p \leftarrow M^i$
    while $|M^p| > 0$ do
        $m_{top} \leftarrow$ highest scored match in $M^p$
        $M^p \leftarrow M^p - \{m_{top}\}$
        for all matches $m_o$ in $M^p$ sorted by descending combined score do
            if $Compatible\left(m_{top}, m_o\right)$ then
                $m_{top} \leftarrow Merge\left(m_{top}, m_o\right)$
                $M^p \leftarrow M^p - \{m_o\}$
            end if
        end for
        $M^f \leftarrow M^f \cup \{m_{top}\}$
    end while
    $M^i \leftarrow M^f$
end for
```

Step 2: Greedy Match Connection. This step takes the grown matches computed in the previous step and further grows them by merging partially compatible matches with low spatial distortion cost. We say that two aligned index matches m_1 and m_2 are partially compatible for merging if both contain non-conflicting additional mappings between query and candidate notes that could be added to the other to form a larger match. This is:

$$PartialComp\left(m_1, m_2\right) = \exists v \in G_1^q \left(v \notin G_2^q \wedge f(v) \notin G_2^i\right) \wedge \exists u \in G_2^q \left(u \notin G_1^q \wedge f(u) \notin G_1^i\right)$$

Two partially compatible aligned matches can be merged if the compatible sub-match of one of the alignments is added to the other match. Here, for aligned matches m_1 and m_2, we define the compatible sub-match $m_s = \left(G_s^q, G_s^i, f_s\right)$ of m_2 as the portion of m_2 that is either absent from or consistent with m_1. A portion of a match considered to be consistent with another if it does not contain conflicting mappings. This is

$$G_s^q = \{v | v \in G_2^q \wedge \left[\left(v \notin G_1^q \wedge f_2\left(v\right) \notin G_1^i\right) \vee \left(v \in G_1^q \wedge f_2\left(v\right) = f_1\left(v\right)\right)\right]\}$$

$$G_s^i = \{f_2\left(v\right) | v \in G_s^q\}$$

$$f_s = G_s^q \rightarrow G_s^i \text{ such that } \forall v \in G_s^q \left(f_s(v) = f_2(v)\right)$$

$$CompSubMatch\left(m_1, m_2\right) = \left(G_s^q, G_s^i, f_s\right)$$

We define a path m_p between aligned matches m_1 and m_2 as a sequence of aligned nodes such that the first alignment in m_p is part of m_1 and the last alignment is part of m_2. Note that the path is itself an aligned match partially compatible with both alignments. If such path exists between matches m_1 and m_2, then a final larger match can be generated by merging m_1 with m_p and then merging the resulting match with the compatible sub-match from m_2.

One drawback of simply connecting matches using paths is that this might lead to final aligned matches that are spatially inconsistent with the layout of the query graph. For this reason, we restrict the greedy matching connection process using a relative spatial distortion cost based on the estimations of translation and scale between query space and index match space based on aligned nodes. Given aligned matches m_1 and m_2, we compute the relative spatial distortion cost of adding m_2 to m_1 as follows: First, let $Loc(x)$ be a function that returns the 2D position of node x on the original image, and let N be an arbitrary set of graph nodes, we define $MeanDist\,(u, N)$ as the function that finds the average distance between the node u and all nodes in N as follows:

$$MeanDist\,(u, N) = \frac{\sum_{v \in N} |Loc(u) - Loc(v)|}{|N|}$$

we then use this function to find the average distance from each node to each other node in G_1^q (query nodes in m_1), and choose a reference node ω with the smallest average distance to all other nodes as the translation reference node:

$$\omega = \underset{v}{\operatorname{argmin}}\, MeanDist\,(v, G_1^q - \{v\})$$

We compute the average distance between the corresponding candidate node $f_1(\omega)$ and each other node in G_1^i, and then we use the average distances of ω and $f_1(\omega)$ to compute a relative scaling factor S_ω from the original query scale to the candidate match scale:

$$S_\omega = \frac{MeanDist\,(\omega, Q_1^q - \{\omega\})}{MeanDist\,\big(f_1\,(\omega)\,, Q_1^i - \{f(\omega)\}\big)}$$

We use the locations of the reference node ω and its corresponding index node $f_1(\omega)$ to compute translations for each space making the aligned reference nodes the new origin. For each query node u in G_2^q from m_2, we simply translate it to its new location relative to the new origin defined by ω:

$$AlignQ\,(u) = Loc\,(u) - Loc\,(\omega)$$

and the corresponding index node of u, $f_2(u)$, in G_2^i from m_2, we use the translation defined by $f_1(\omega)$ and apply the scaling factor to compute the new locations of these vertices in the new space:

$$AlignI\,(u) = [Loc\,(f_2\,(u)) - Loc\,(f_1(\omega))]\,S_\omega$$

Finally, we compute the relative spatial distortion cost *SpatialCost* as the average of the normalized distances between each projected query node location from m_2 and their corresponding index node locations:

$$SpatialCost\,(m_1, m_2) = \frac{\sum_{u \in G_2^q} |AlignQ\,(u) - AlignI\,(u)|}{MeanDist\,\big(f_1\,(\omega)\,, Q_1^i - \{f(\omega)\}\big)}$$

The greedy match connection procedure is as follows. For each binary image that has at least one match found by the first layer, take the match m_{top} with the current top score and start a greedy connection procedure. Find the next partially compatible match m_o with spatial alignment cost below lower than a threshold $MaxSpatialCost = 3.0$. Run a Breadth First Search (BFS) with limited depth of search $MaxSearchDepth = 4$ to find the shortest path m_{path1} from any alignment in the largest match to any compatible alignment in the smaller match. If such path is found, then we repeat the search procedure starting from the smallest match to the largest to find path m_{path2}. We defined the extended matches $m_{e1} = Merge\,(m_{top}, m_{path1})$, and $m_{e2} = Merge\,(m_o, m_{path2})$ as the aligned matches that contain the original match along with the path that connects to the other match. If $m_{path1} = m_{path2}$, we compute the final connected match m_{c1} as $m_{c1} = \mathit{Merge}\left(m_{e1}, \mathit{CompSubMatch}\left(m_{e1}, m_o\right)\right)$. If $m_{path1} \neq m_{path2}$, then we also compute the alternative match m_{c2} as $m_{c2} = \mathit{Merge}\left(m_{e2}, \mathit{CompSubMatch}\left(m_{e2}, m_{top}\right)\right)$, choose to replace m_{top} with the match with the highest score between m_{c1} and m_{c2}. We repeat this procedure until no further partially compatible matches are found for m_{top}, and then we repeat the procedure for the next highest scored match until all matches have been considered.

Step 3: Filter incompatible matches. For each binary image matched, greedily choose a subset of matches such that a single candidate node is aligned to at most one match of the final set of matches. A single node of the query, however, might still be aligned to multiple locations on a given binary image.

The procedure is as follows: For each binary image with aligned matches, the aligned matches are sorted by descending score, then mark all nodes in the binary image graph as unmatched. Starting from the top match, we check for each match m_x if all the index nodes G_x^i in m_x are still marked as unmatched, and if this is true then we accept m_x as part of the final set, and we mark the corresponding index nodes in G_x^i as matched, otherwise we simply discard the match.

Step 4: Group matches across binary images. This step reduces the final set of candidates by grouping trivially identical matches across related binary images. For lecture video key-frames, this step will group matches across contiguous key-frames of the same video that have an area overlap above $\mathit{MinAreaOverlap} = 50\%$. If the binary images in the index are rendered LaTeXexpressions extracted from lecture notes, this step with group matches across binary images associated with the same LaTeXstring.

6.2 Experiments

To test the effectiveness of the proposed method, we use data from the AccessMath dataset. The AccessMath contains 20 lecture videos recorded using a still camera in the classroom. The resolution of each lecture video is 1080p. Out of the 20 lecture videos, 13 of them are accompanied by auxiliary lecture notes provided in LaTeXformat. We experiment on this subset of videos that have auxiliary notes describing most of the math expressions found in the video.

For training of handwritten math symbol classifiers, we use handwritten math expressions from the CROHME 2016 dataset [77]. In this case, we expand the dataset by adding the testing dataset for the matrix recognition subtask in order to improve recognition results for matrices given that the current evaluation videos are from linear algebra lectures. To reduce the index size and to improve retrieval performance, we group together classes with very similar shapes, and this reduces the original 101 classes in CROHME to just 91 classes. In addition, we train one math symbol classifier per video based on the expected set of classes for that lecture as described in Section 6.1.1 which further reduces the number of classes to no more than 50 per lecture video.

For training of typeset math symbol classifiers, we generate artificial data as described in Section 5.3. We generate a 1000 LaTeXimages for each isolated symbol class (91 classes) using distorted copies of the same characters drawn using different fonts, for a total of 91,000 training symbols. We also use one math symbol classifier per lecture for typeset math symbols just as is done for the handwritten symbols.

After lecture video preprocessing, a total of 219 key-frames binary are indexed from the 13 videos defining 2170 symbol pairs index entries, and 715,989 pair instances in total. A total of 1118 rendered LaTeXmath expressions from 782 unique LaTeXstrings are indexed on a parallel index for the lecture notes, which has 1696 symbol pairs index entries, with 53,684 pair instances in total.

$= \vec{0}$
Query-01

$x + 2y = 3$
Query-02

$\begin{bmatrix} -2 \\ 3 \end{bmatrix}$
Query-03

$2z + w$
Query-04

$(1/3, 4/3)$
Query-05

$= \begin{bmatrix} a \\ b \end{bmatrix}$
Query-06

$\vec{v_2}$
Query-07

$V = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} : x, y \in \mathbb{R} \right\}$
Query-08

$b_1 = b_2/3$
Query-09

$a, b \in \mathbb{R}$
Query-10

$\vec{0} \in S$
Query-11

$M_b(\mathbb{R})$
Query-12

$\begin{bmatrix} 1 & 0 & \frac{-5}{19} \\ 0 & 1 & \frac{27}{19} \end{bmatrix}$
Query-13

$r(A)$
Query-14

$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 1 & 1 \end{bmatrix} \xrightarrow{R_2 \circlearrowright R_3} \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 3 \end{bmatrix}$
Query-15

$\begin{pmatrix} \pi \\ e \end{pmatrix}$
Query-16

$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 0 & 7 \\ 1 & 2 \end{pmatrix} \neq \begin{pmatrix} 0 & 7 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
Query-17

$\vec{u} \in V$
Query-18

$f(x)$
Query-19

$= \mathbf{A}^T + \mathbf{A}$
Query-20

Figure 6.3: Typeset LaTeX Evaluation Queries.

The lecture video index takes 1.88 GB on disk while the lecture notes index require 1.33 GB.

We pseudo-randomly choose 20 unique queries for evaluation based on the following criteria. First, each query is randomly selected from the rendered LaTeXexpressions from the lecture notes. Second, the query must also appear on at least one key-frame of the corresponding lecture video with no more than 1 symbol of difference. Third, the query must have at least 3 symbols. Fourth, the query is not a sub-expression of any other query previously selected. Finally, the sampled query does not completely contain any other query previously selected. The 20 rendered LaTeXexpressions selected are used as typeset LaTeXqueries and their corresponding handwritten versions extracted from the binary key-frame images. The final typeset queries from rendered LaTeXexpressions are shown in Figure 6.3 and their corresponding handwritten versions are shown in Figure 6.4.

We used the selected queries to evaluate the two main retrieval modalities: handwritten queries for handwritten key-frames search (Handwriting-to-Handwriting), typeset LaTeXqueries for typeset LaTeXimages search (LaTeX-to-LaTeX). Then, we also evaluate the search across modalities (cross-modal search): handwritten queries for typeset LaTeXimages search (Handwriting-to-LaTeX), and typeset LaTeXqueries for handwritten key-frames search (LaTeX-to-Handwriting). A total of four modalities for image-based search are evaluated.

We evaluate the performance of the system for each modality based on recall and mean reciprocal rank of the target matches. A target match is an identical match of the math expression in the query. For a given query q in the evaluation set Q, we define the rank r as the position in the ranking at which the query target was found, if it was found at all. We define the reciprocal rank (RR) of q as follows:

$$RR = \begin{cases} \frac{1}{r} & \textit{if } 1 \leq r \leq r_{max} \\ 0 & \textit{otherwise} \end{cases}$$

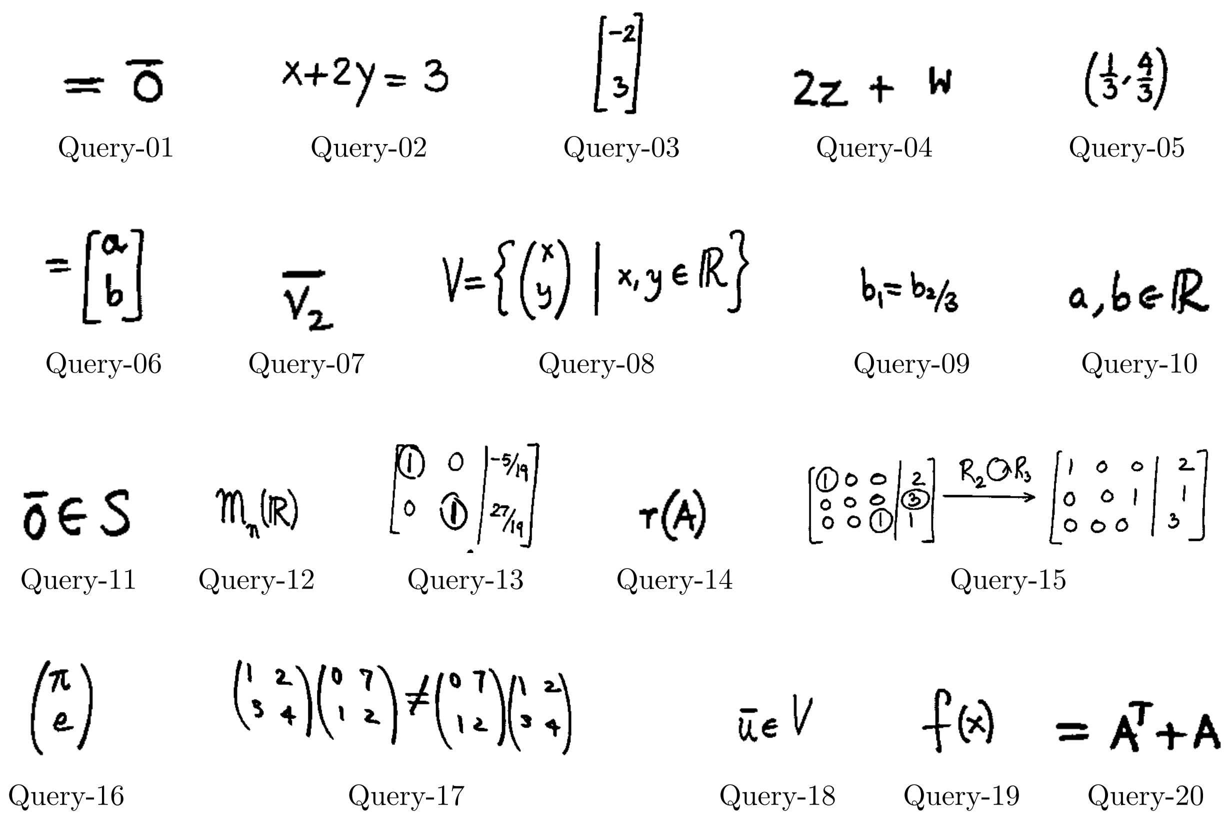


Figure 6.4: Handwritten Evaluation Queries.

Table 6.1: Recall and MRR Results for each search modality

Modality	Recall	MRR
Handwriting-to-Handwriting	95%	0.9250
LaTeX-to-LaTeX	100%	0.9208
Handwriting-to-LaTeX	85%	0.5592
LaTeX-to-Handwriting	95%	0.8265

where $r_{max} = 20$ as we only consider the first top-20 matches per query. We then define the mean reciprocal rank (*MRR*) as the mean of *RR* for all queries in the evaluation set Q. We then define recall of target matches as the percentage of queries from the evaluation set for which our method found the target match within the first top-20 results.

For typeset LaTeXimages search, formulas are well delimited and our definition of a target match becomes very strict by only accepting formulas that are identical to the query without missing or extra elements, except for minor symbol replacement that do not change the meaning of the match (E.g. $\bar{v}$ matches $\vec{v}$). For handwritten key-frames search, the math formulas are not delimited at all, and we simply accept any match that contains all elements of the query, with minor replacements as well, as a valid match while ignoring the context in which it was found.

Evaluation results for the four search modalities are shown in Table 6.1. Only the typeset to typeset search modality (LTX-to-LTX) achieved perfect recall meaning that the target was always found within the top-20 results. This is not surprising since the typeset images are very regular and

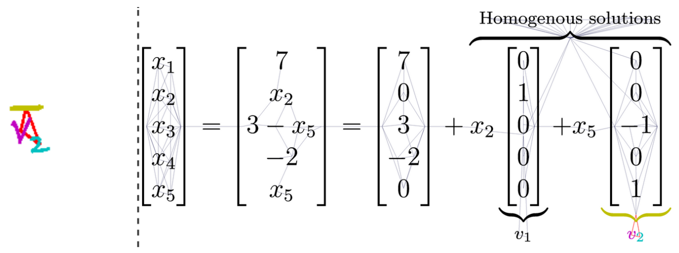


Figure 6.5: Top-1 match for Query-07 in HW-to-LTX modality. The horizontal curly brace is matched to the horizontal bar from the query despite of the notorious difference in size between these symbols, and overall the match gets ranked too high.

both classification of symbols and LOS-graph structures are very consistent for similar expressions and sub-expressions. On the other hand, modalities involving handwritten queries or handwritten key-frame images had a higher number of classification errors and inconsistency in graph structures causing some targets to be completely missing from the top-20. However, it was noticed that despite the existing errors, the system was able to find most targets as long as at least one small portion of the query was consistently matched with a candidate index image.

We can observe that the highest MRR metrics are obtained by the same-modality search (HW-to-HW, LTX-to-LTX), and as expected cross-modality search results become less accurate, especially when handwritten queries are used to find typeset LaTeXimages. Again, this is mostly due to the differences in consistency of representations between modalities. For this reason, even the search between handwritten queries and handwritten key-frames achieves a higher MRR than searching the same key-frames using typeset queries. Figure 6.6 shows the top-5 results for query 10 from the evaluation set. It can bee seen that the target match is found at the top-1 location for both cases of same-modality search. However, search across modalities is harder and other irrelevant matches can be ranked higher than the target match.

In a sense, a misclassified handwritten shape might be consistently associated with the same wrong label across handwritten images, making it possible for queries with the same classification errors to match it. On the other hand, a more regular typeset image with lower classification errors has a lower chance of matching these erroneously classified handwritten symbols.

Typeset graphs are also very regular and consistent for the same sub-expressions while handwritten math expressions usually have more local deformations and variations in shape and size for multiple instances of the same sub-expression. In addition, the current matching algorithm does not consider size ratios of the pairs of symbols being matched, and it is very common to find pairs where symbols with similar shapes are matched, but the difference in size ratios is too large to consider it a good match. An example is shown in Figure 6.5 where the horizontal bar above v_2 is being matched to a horizontal curly brace and the whole match ranks higher than other images that are actually more visually similar to the query. Also, the same example shows how the current model might rank certain matches very high independently of the content in which they appear. Sometimes, the context of a match can make it more or less relevant than other identical matches that appear on different contexts. In particular for the case of Query-07 in the HW-to-LTX modality, an exact match for the query is found at rank 8, and this exact match should have been ranked on top instead of the current top match shown in Figure 6.5

For all the reasons mentioned before, the hardest modality of search becomes using handwritten queries to match typeset regions, as confirmed by this modality having the lowest MRR in Table

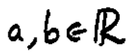
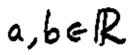
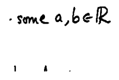
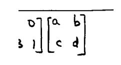
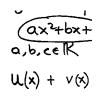
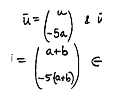
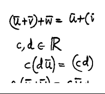
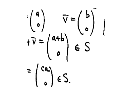
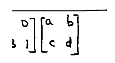
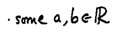
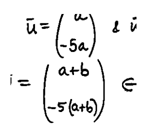
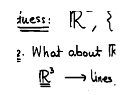
	Search Modalities			
	HW-TO-HW	LTX-to-LTX	HW-to-LTX	LTX-to-HW
Query		$a, b \in \mathbb{R}$		$a, b \in \mathbb{R}$
Rank	Results			
1		$a, b \in \mathbb{R}$	$a, b, c \in \mathbb{R}$	
2		$\vec{b} \in \mathbb{R}^n$	$a, b \in \mathbb{R}$	
3		$a, b, c \in \mathbb{R}$	$= \begin{bmatrix} 1 & 0 \\ -3 & 1 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$	
4		$c, d \in \mathbb{R}$	$= \begin{bmatrix} a & b \\ c & d \end{bmatrix}$	
5		$S = \{v_1, v_2, \ldots, v_k\} \in \mathbb{R}^n$	$\left.\begin{matrix} c_1 + c_2 + 3c_3 = 0 \\ c_2 + c_3 = 0 \end{matrix}\right\} \quad c_3 \rightarrow \text{free variable}$	

Figure 6.6: Query 10, Top-5 results for all modalities

6.1. Handwritten queries might have irregularities, classification errors, and lower confidences for top labels, and they are used to match very regular images where such errors are rare. On the other hand, we have defined a more strict matching criteria for the typeset LaTeX index, and we observed that many handwritten queries actually find relevant subexpressions or even larger expressions containing the entire query.

Note that typeset images can be constructed for symbolic representations meaning that using a typeset LaTeX image query is equivalent to start with the symbolic LaTeX representation and execute symbolic-to-image search by rendering the query. Also, using handwritten images to find rendered LaTeX images can permit image-to-symbolic search since the symbolic representation of the matched imaged is known, and it is also possible to render all formulas in a symbolic dataset and index them as images for search using the proposed method.

6.3 Summary

In this chapter, we presented a novel method for retrieval of mathematical notation from images extracted from lecture videos or rendered LaTeXmath expressions. The proposed method is a generalization of the retrieval methods used for symbolic math information retrieval described in Chapter 3.

Our initial study shows that current results for search are promising and that in the future it might be possible to obtain comparable results to symbolic approaches in the image domain.

However, some questions remain open like what would be a useful way to incorporate search functionalities like unification of symbols and wildcard matching to the image-based domains.

We also shown that cross-modality search is possible, by creating a bridge between the symbolic and the image domains by rendering the symbolic math expressions to images. The alternative path is to use a full math expression recognition approach to convert images to the symbolic domain, but in general, rendering symbolic expressions is easier and more accurate than recognizing the complete structure of math expressions from images.

In future versions of our model we would like to consider size ratios between symbol pairs being matched, to avoid scoring pairs with inconsistent size ratios too high. Also, we will need some adaptation of precision for a match or some other similarity metrics that considers the context of a match, and prefers exact matches that are isolated to the ones surrounded by many extra symbols.

Chapter 7

Conclusion

In this work we have presented different models for retrieval of mathematical notation from documents and images. Different contributions have been made, not just toward solving the mathematical notation retrieval problem, but also for solving some of the required preprocessing steps like math symbol recognition and lecture video summarization.

Through all of our research work we could observe that simple models based on visual appearance tend to be enough to produce satisfactory results for mathematical notation search as shown in Chapters 3 and 6. However, as we observed in our experiments using semantic representations like Operator Trees (Chapter 3), improved results can be obtained if both visual and semantic representations are combined for search.

Consistently in all three formula representations used in this work (SLT, OPT and LOS graphs), we could observed that matching symbol pairs between queries and candidates is enough to find relevant matches in a fast and scalable manner.

We have also shown that initial results from simple pair matching can be improved if structural matching and re-scoring algorithms are applied to the initial sets of candidates. Using a multi-layer retrieval model allows us to quickly identify the most likely relevant matches and filter less promising candidates before using more computationally expensive methods. This makes the search process more accurate but significantly slower. In all cases, when exact matches exists for given queries, they will usually be found at the top of the list after the first layer, and this means that if the user is not interested in partial matches, we could simply return results from the first layer and have these relevant matches retrieved at the top. An exception is the image-based matching where the lack of consideration for precision of a match can cause exact matches found as sub-expressions of larger math expressions to be ranked as high as identical matches.

If relevance assessments are available, we can further improve our search results in all cases by training regressors (see Chapter 3) that learn to predict relevance for a match using a set of similarity metrics as input features. We expect that the best retrieval results could be obtained if the selected similarity metrics cover the most important aspects of what makes a match relevant to a query, and a large enough dataset of relevance assessments for large query sets are available for training.

7.1 Contributions

In terms of symbolic formula retrieval (Chapter 3), we proposed the novel *Maximum Subtree Similarity (MSS)* metric for ranking formulae along with methods for unification and wildcard matching, and we achieved state-of-the-art performance on the NTCIR-11 and NTCIR-12 formula retrieval benchmarks, indicating that our approach is efficient in terms of time and space, and produces effective formula search results. Visual and semantic representations can be combined using the proposed method in the Tangent-S model, and improved search results can be obtain in this manner. The source code of our symbolic formula retrieval approach is publicly available[1].

In Chapter 4, we contribute a novel method for lecture video summarization based on analysis of conflicts between stable connected components from binarized video frames, allowing us to extract the handwritten whiteboard content from the videos. We contribute with an spatio-temporal structure that stores detailed temporal information of the handwritten content on the video. Using this structure, we have created a navigation tool that lets the users find the portion of the video where a particular connected component is being written.

We also provide a new dataset with 12 fully labeled lecture videos that can be used for benchmarking of approaches for binarization and summarization of lecture videos in the future. It will be made publicly available in the near future along with the tools used for labeling the videos.

In Chapter 5, we contribute a shape-based feature set for recognition of on-line and off-line handwritten math symbols along with a method for synthetic data generation for math symbol recognition. The code for the math symbol recognizer and the synthetic data generation model have been made publicly available[2].

Finally, in Chapter 6 we proposed a novel approach for retrieval of mathematical notation from images that works with probabilistic math symbol recognition and does not require parsing the structure of math expressions. We have provided preliminary results that show that cross-modal search between images of symbolic expressions and a handwritten expression image index (and vice-versa) is possible using this method.

7.2 Future Work

In general, better similarity metrics are still required that would let us produce rankings that better approximate perceived relevance for users. Larger-scale human experiments are needed to identify which features affect the perception of formula similarity, and how different populations identify formulae as similar (e.g., for mathematical experts vs. non-experts).

We would also like to consider interactions between formulas and their context as we have only studied here isolated formulas, but the context in which a match appears is typically what define if the match will be an useful result for a given query. Interactions between math expressions and formulas need to be considered and also interactions between formulas and other formulas in the documents.

We would like to consider further extensions for the image-based model Tangent-V by adapting unification and wildcard matching process to the visual domain. Unification in this context would consider query shapes that are consistently aligned with other shapes from the index, and would increase the score of partial matches were a re-labeling of index nodes makes them more similar to the original query graph. For wildcards, the biggest challenge is how to define the wildcard expansion procedure in a visual domain where the math expressions are not delimited and multiple matches can be found on a single graph.

For our lecture video summarization approach, we would like to test the effectiveness of this method using blackboard/chalkboard videos. We would also like to further extend the proposed

[1]`https://cs.rit.edu/~dprl/Software.html\#tangent`

[2]`https://cs.rit.edu/~dprl/Software.html\#crohme`

method to work on harder cases where there is camera zooming/panning and lectures recorded using multiple cameras.

Finally, in terms of math symbol recognition, we would like to consider other techniques that have shown more accurate results than methods purely based on shapes. In particular, we would like to explore methods related to convolutional neural networks for shape-based classification and complementary recurrent neural networks for analysis of the temporal sequences defined by the symbol traces.

Bibliography

[1] John Adcock, Matthew Cooper, Laurent Denoue, Hamed Pirsiavash, and Lawrence A Rowe. Talkminer: A lecture webcast search engine. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 241–250. ACM, 2010.

[2] Akiko Aizawa, Michael Kohlhase, and Iadh Ounis. NTCIR-10 math pilot task overview. In *Proc. of the 10th NTCIR Conference, Tokyo, Japan*, pages 654–661, 2013.

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

[4] Moody Ebrahem Altamimi and Abdou Youssef. A math query language with an expanded set of wildcards. *Mathematics in Computer Science*, 2(2):305–331, 2008.

[5] Francisco Alvaro, Joan-Andreu Sánchez, and Jose-Miguel Benedi. Classification of on-line mathematical symbols with hybrid features and recurrent neural networks. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1012–1016. IEEE, 2013.

[6] Francisco Álvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. Recognition of on-line handwritten mathematical expressions using 2D stochastic context-free grammars and hidden markov models. *Pattern Recognition Letters*, 35:58–67, 2014.

[7] Relja Arandjelović and Andrew Zisserman. Three things everyone should know to improve object retrieval. In *CVPR*, pages 2911–2918. IEEE, 2012.

[8] Yannis Avrithis and Giorgos Tolias. Hough pyramid matching: Speeded-up geometry re-ranking for large scale image retrieval. *International Journal of Computer Vision*, 107(1):1–19, 2014.

[9] Purnendu Banerjee, Ujjwal Bhattacharya, and Bidyut B Chaudhuri. Automatic detection of handwritten texts from video frames of lectures. In *International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 627–632. IEEE, 2014.

[10] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006.

[11] Jon Louis Bentley. An almost optimal algorithm for unbounded searching. *Inf. Process. Lett.*, 5(3):82–87, 1976.

[12] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[13] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, 2003.

[14] Yang Cao, Changhu Wang, Liqing Zhang, and Lei Zhang. Edgel index for large-scale sketch-based image search. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 761–768. IEEE, 2011.

[15] Edwin Catmull and Raphael Rom. A class of local interpolating splines. *Computer Aided Geometric Design*, 74:317–326, 1974.

[16] Hyun Sung Chang, Sanghoon Sull, and Sang Uk Lee. Efficient video indexing scheme for content-based retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1269–1279, Dec 1999.

[17] Houssem Chatbri, Kenny Davila, Keisuke Kameyama, and Richard Zanibbi. Shape matching using keypoints extracted from both the foreground and the background of binary images. In *IPTA*, pages 205–210. IEEE, 2015.

[18] Houssem Chatbri, Paul Kwan, and Keisuke Kameyama. An application-independent and segmentation-free approach for spotting queries in document images. In *ICPR*, pages 2891–2896. IEEE, 2014.

[19] Chekuri Choudary and Tiecheng Liu. Extracting content from instructional videos by statistical modelling and classification. *Pattern analysis and applications*, 10(2):69–81, 2007.

[20] Chekuri Choudary and Tiecheng Liu. Summarization of visual content in instructional videos. *IEEE Transactions on Multimedia*, 9(7):1443–1455, 2007.

[21] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[22] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[23] Kenny Davila. Appearance-based retrieval of mathematical notation in documents and lecture videos. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 1165–1165. ACM, 2016.

[24] Kenny Davila, Anurag Agarwal, Roger Gaborski, Richard Zanibbi, and Stephanie Ludi. Accessmath: Indexing and retrieving video segments containing math expressions based on visual similarity. In *Image Processing Workshop (WNYIPW), 2013 IEEE Western New York*, pages 14–17. IEEE, 2013.

[25] Kenny Davila, Stephanie Ludi, and Richard Zanibbi. Using off-line features and synthetic data for on-line handwritten math symbol recognition. In *ICFHR*, pages 323–328. IEEE, 2014.

[26] Kenny Davila and Richard Zanibbi. Layout and semantics: Combining representations for mathematical formula search. *SIGIR*, 2017.

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

[28] Adrien Delaye and Eric Anquetil. Hbf49 feature set: A first unified baseline for online symbol recognition. *Pattern Recognition*, 46(1):117–130, 2013.

[29] Paul E Dickson, W Richards Adrion, and Allen R Hanson. Whiteboard content extraction and analysis for the classroom environment. In *IEEE International Symposium on Multimedia*, pages 702–707. IEEE, 2008.

[30] Markus Eberts, Adrian Ulges, and Ulrich Schwanecke. Amigo-automatic indexing of lecture footage. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1206–1210. IEEE, 2015.

[31] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jurgen Hafner, Denis Lee, Dragutin Petkovic, et al. Query by image and video content: The QBIC system. *Computer*, 28(9):23–32, 1995.

[32] Liangcai Gao, Yuehan Wang, Leipeng Hao, and Zhi Tang. ICST math retrieval system for NTCIR-11 math-2 task. In *NTCIR*. Citeseer, 2014.

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

[34] Utpal Garain and Bidyut Baran Chaudhuri. Recognition of online handwritten mathematical expressions. *Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(6):2366–2376, 2004.

[35] Zhiwei Guan and Edward Cutrell. An eye tracking study of the effect of target rank on web search. In *Proc. ACM SIGCHI*, pages 417–420, 2007.

[36] Ferruccio Guidi and Claudio Sacerdoti Coen. A survey on retrieval of mathematical knowledge. In *Intelligent Computer Mathematics*, pages 296–315. Springer, 2015.

[37] Radu Hambasan, Michael Kohlhase, and Corneliu-Claudiu Prodescu. MathWebSearch at NTCIR-11. In *NTCIR*. Citeseer, 2014.

[38] H. Hiroya and H. Saito. Partial-match retrieval with structure-reflected indices at the NTCIR-10 math task. In *Proc. NTCIR-10*, pages 692–695, 2013.

[39] Lei Hu and Richard Zanibbi. HMM-based recognition of online handwritten mathematical symbols using segmental k-means initialization and a modified pen-up/down feature. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 457–462. IEEE, 2011.

[40] Lei Hu and Richard Zanibbi. Line-of-sight stroke graphs and parzen shape context features for handwritten math formula representation and symbol segmentation. In *ICFHR*, pages 180–186. IEEE, 2016.

[41] Weiming Hu, Nianhua Xie, Li Li, Xianglin Zeng, and Stephen Maybank. A survey on visual content-based video indexing and retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(6):797–819, 2011.

[42] Bing Quan Huang, YB Zhang, and M-T Kechadi. Preprocessing techniques for online handwriting recognition. In *Intelligent Text Categorization and Clustering*, pages 25–45. Springer, 2009.

[43] Ali Shariq Imran, Sukalpa Chanda, Faouzi Alaya Cheikh, Katrin Franke, and Umapada Pal. Cursive handwritten segmentation and recognition for instructional videos. In *International Conference on Signal Image Technology and Internet Based Systems (SITIS)*, pages 155–160. IEEE, 2012.

[44] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. On the burstiness of visual elements. In *CVPR*, pages 1169–1176. IEEE, 2009.

[45] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Improving bag-of-features for large scale image search. *International Journal of Computer Vision*, 87(3):316–336, 2010.

[46] Shahab Kamali and Frank Wm Tompa. A new mathematics retrieval system. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1413–1416. ACM, 2010.

[47] Shahab Kamali and Frank Wm Tompa. Structural similarity search for mathematics retrieval. In *Intelligent Computer Mathematics*, pages 246–262. Springer, 2013.

[48] Toshihiro Kanahori and Masakazu Suzuki. Refinement of digitized documents through recognition of mathematical formulae. In *DIAL*, pages 6–pp. IEEE, 2006.

[49] Michael Kohlhase, Bogdan A Matican, and Corneliu-Claudiu Prodescu. MathWebSearch 0.5: Scaling an open formula search engine. In *Intelligent Computer Mathematics*, pages 342–357. Springer, 2012.

[50] Michael Kohlhase and Ioan Sucan. A search engine for mathematical formulae. In *LNCS Vol. 4120*, pages 241–253. Springer, 2006.

[51] Giovanni Yoko Kristianto, G Topić, and A Aizawa. The MCAT math retrieval system for NTCIR-12 MathIR task. In *Proc. NTCIR-12*, pages 323–330, 2016.

[52] Giovanni Yoko Kristianto, Goran Topić, Florence Ho, and Akiko Aizawa. The MCAT math retrieval system for NTCIR-11 Math Track. In *NTCIR*, 2014.

[53] Greg C Lee, Fu-Hao Yeh, Ying-Ju Chen, and Tao-Ku Chang. Robust handwriting extraction and lecture video summarization. *Multimedia Tools and Applications*, pages 1–19, 2016.

[54] Kai Li, Jue Wang, Haoqian Wang, and Qionghai Dai. Structuring lecture videos by automatic projection screen localization and analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(6):1233–1246, 2015.

[55] Xinchao Li, Martha Larson, and Alan Hanjalic. Pairwise geometric matching for large-scale object retrieval. In *CVPR*, pages 5153–5161, June 2015.

[56] Zhu Li, Guido M Schuster, and Aggelos K Katsaggelos. Minmax optimal video summarization. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(10):1245–1256, 2005.

[57] Shuang Liang and Zhengxing Sun. Sketch retrieval and relevance feedback with biased SVM classification. *Pattern Recognition Letters*, 29(12):1733–1741, 2008.

[58] Xiaoyan Lin, Liangcai Gao, Xuan Hu, Zhi Tang, Yingnan Xiao, and Xiaozhong Liu. A mathematics retrieval system for formulae in layout presentations. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 697–706. ACM, 2014.

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

[60] Aldo Lipani, Linda Andersson, Florina Piroi, Mihai Lupu, and Allan Hanbury. TUW-IMP at the NTCIR-11 Math-2. In *NTCIR*, 2014.

[61] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[62] Tiecheng Liu and Chekuri Choudary. Content extraction and summarization of instructional videos. In *2006 International Conference on Image Processing*, pages 149–152. IEEE, 2006.

[63] Tiecheng Liu and John R Kender. Rule-based semantic summarization of instructional videos. In *International Conference on Image Processing*, volume 1, pages I–601. IEEE, 2002.

[64] Tiecheng Liu and John R Kender. Semantic mosaic for indexing and compressing instructional videos. In *International Conference on Image Processing*, volume 1, pages I–921. IEEE, 2003.

[65] David G Lowe. Object recognition from local scale-invariant features. In *IEEE International conference on Computer Vision*, volume 2, pages 1150–1157. IEEE, 1999.

[66] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[67] Scott MacLean and George Labahn. Elastic matching in linear time and constant space. In *International Workshop on Document Analysis Systems*, pages 551–554, 2010.

[68] Scott MacLean, George Labahn, Edward Lank, Mirette Marzouk, and David Tausky. Grammar-based techniques for creating ground-truthed sketch corpora. *International Journal on Document Analysis and Recognition (IJDAR)*, 14(1):65–74, 2011.

[69] Simone Marinai, Beatrice Miotti, and Giovanni Soda. Mathematical symbol indexing using topologically ordered clusters of shape contexts. In *ICDAR*, pages 1041–1045. IEEE, 2009.

[70] Simone Marinai, Beatrice Miotti, and Giovanni Soda. Using earth mover's distance in the bag-of-visual-words model for mathematical symbol retrieval. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1309–1313. IEEE, 2011.

[71] Bruce R Miller and Abdou Youssef. Technical aspects of the digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence*, 38(1-3):121–136, 2003.

[72] Jozef Mišutka and Leo Galamboš. System description: Egomath2 as a tool for mathematical searching on wikipedia. org. In *Intelligent Computer Mathematics*, pages 307–309. Springer, 2011.

[73] Arthur G Money and Harry Agius. Video summarisation: A conceptual framework and survey of the state of the art. *Journal of Visual Communication and Image Representation*, 19(2):121–143, 2008.

[74] Harold Mouchère, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain. Crohme2011: Competition on recognition of online handwritten mathematical expressions. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1497–1500. IEEE, 2011.

[75] Harold Mouchère, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain. ICFHR 2012 competition on recognition of on-line mathematical expressions (crohme 2012). In *International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 811–816. IEEE, 2012.

[76] Harold Mouchere, Christian Viard-Gaudin, Richard Zanibbi, and Utpal Garain. ICFHR 2014 competition on recognition of on-line handwritten mathematical expressions (CROHME 2014). In *International Conference on Frontiers in handwriting recognition (ICFHR)*, pages 791–796. IEEE, 2014.

[77] Harold Mouchère, Christian Viard-Gaudin, Richard Zanibbi, and Utpal Garain. ICFHR 2016 CROHME: Competition on recognition of online handwritten mathematical expressions. In *International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016.

[78] Harold Mouchere, Christian Viard-Gaudin, Richard Zanibbi, Utpal Garain, Dae Hwan Kim, and Jin Hyung Kim. ICDAR 2013 CROHME: Third international competition on recognition of online handwritten mathematical expressions. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1428–1432. IEEE, 2013.

[79] Harold Mouchère, Richard Zanibbi, Utpal Garain, and Christian Viard-Gaudin. Advancing the state of the art for handwritten math recognition: the CROHME competitions, 2011–2014. *International Journal on Document Analysis and Recognition (IJDAR)*, 19(2):173–189, 2016.

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

[81] Cuong Nguyen, Yuzhen Niu, and Feng Liu. Video summagator: An interface for video summarization and navigation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 647–650. ACM, 2012.

[82] Tam T Nguyen, Kuiyu Chang, and Siu Cheung Hui. A math-aware search engine for math question answering system. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 724–733. ACM, 2012.

[83] Tam T Nguyen, Siu Cheung Hui, and Kuiyu Chang. A lattice-based approach for mathematical search using formal concept analysis. *Expert Systems with Applications*, 39(5):5820–5828, 2012.

[84] Wayne Niblack. *An introduction to digital image processing.* Strandberg Publishing Company, 1985.

[85] Shunsuke Ohashi, Giovanni Yoko Kristianto, Goran Topić, and Akiko Aizawa. Efficient algorithm for math formula semantic search. *IEICE Transactions on Information and Systems*, 99(4):979–988, 2016.

[86] Masaki Onishi, Masao Izumi, and Kunio Fukunaga. Blackboard segmentation using video image of lecture and its applications. In *International Conference on Pattern Recognition*, volume 4, pages 615–618. IEEE, 2000.

[87] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[88] Nidhin Pattaniyil and Richard Zanibbi. Combining TF-IDF text retrieval with an inverted index over symbol pairs in math expressions: The Tangent math search engine at NTCIR 2014. *Proc. 1tth NII Testbeds and Community for Information Access Research (NTCIR), Tokyo, Japan*, 2014.

[89] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[90] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, 1985.

[91] James Philbin, Ondřej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, pages 1–8. IEEE, 2007.

[92] James Philbin, Ondřrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, pages 1–8. IEEE, 2008.

[93] José María González Pinto, Simon Barthel, and Wolf-Tilo Balke. QUALIBETA at the NTCIR-11 Math 2 task: An attempt to query math collections. In *NTCIR*, 2014.

[94] Réjean Plamondon, Christian OReilly, Javier Galbally, Abdullah Almaksour, and Éric Anquetil. Recent developments in the study of rapid human movements with the kinematic theory: Applications to handwriting and signature synthesis. *Pattern Recognition Letters*, 35:225–235, 2014.

[95] Thomas Plötz, Christian Thurau, and Gernot A Fink. Camera-based whiteboard reading: New approaches to a challenging task. In *International Conference on Frontiers in Handwriting Recognition*, pages 385–390, 2008.

[96] John Ross Quinlan. *C4.5: Programs for Machine Learning*, volume 1. Morgan Kaufmann, 1993.

[97] Tony M Rath and Rudrapatna Manmatha. Word spotting for historical documents. *IJDAR*, 9(2-4):139–152, 2007.

[98] Stephan Repp and Christoph Meinel. Segmentation of lecture videos based on spontaneous speech recognition. In *Multimedia, 2008. ISM 2008. Tenth IEEE International Symposium on*, pages 692–697. IEEE, 2008.

[99] M Rûzicka, Petr Sojka, and M Lıška. Math indexer and searcher under the hood: Fine-tuning query expansion and unification strategies. In *Proc. NTCIR-12*, pages 331–337, 2016.

[100] Michal Ružicka, Petr Sojka, and Martin Líška. Math indexer and searcher under the hood: History and development of a winning strategy. In *Proc. of the 11th NTCIR Conference on Evaluation of Information Access Technologies*, 2014.

[101] Prateek Sarkar and George Nagy. Style consistent classification of isogenous patterns. *Transactions on Pattern Analysis and Machine Intelligence*, 27(1):88–98, 2005.

[102] Christopher Sasarak, Kevin Hart, Richard Pospesel, David Stalnaker, Lei Hu, Robert Livolsi, Siyu Zhu, and Richard Zanibbi. min: A multimodal web interface for math search. In *Symp. Human-Computer Interaction and Information Retrieval, Cambridge, MA*. Citeseer, 2012.

[103] Robert E Schapire and Yoav Freund. *Boosting: Foundations and Algorithms.* The MIT Press, 2012.

[104] Thomas Schellenberg, Bo Yuan, and Richard Zanibbi. Layout-based substitution tree indexing and retrieval for mathematical expressions. In *IS&T/SPIE Electronic Imaging*, pages 82970I–82970I. International Society for Optics and Photonics, 2012.

[105] M. Schubotz. Challenges of mathematical information retrieval in the NTCIR-11 Math Wikipedia Task. In *SIGIR*, pages 951–954, 2015.

[106] Moritz Schubotz, Abdou Youssef, Volker Markl, Howard S Cohl, and Jimmy J Li. Evaluation of similarity-measure factors for formulae based on the NTCIR-11 Math Task. In *NTCIR*, 2014.

[107] Rajiv Ratn Shah, Yi Yu, Anwar Dilawar Shaikh, Suhua Tang, and Roger Zimmermann. Atlas: automatic temporal segmentation and annotation of lecture videos based on modelling transition time. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 209–212. ACM, 2014.

[108] Patrice Simard, David Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[109] Josef Sivic and Andrew Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, pages 1470–1477. IEEE, 2003.

[110] Petr Sojka and Martin Líška. Indexing and searching mathematics in digital libraries. In *Intelligent Computer Mathematics*, pages 228–243. Springer, 2011.

[111] Pedro Sousa and Manuel J Fonseca. Sketch-based retrieval of drawings using spatial proximity. *Journal of Visual Languages & Computing*, 21(2):69–80, 2010.

[112] David Stalnaker and Richard Zanibbi. Math expression retrieval using an inverted index over symbol pairs. In *IS&T/SPIE Electronic Imaging*, pages 940207–940207. International Society for Optics and Photonics, 2015.

[113] C Sujatha and Uma Mudenagudi. A study on keyframe extraction methods for video summary. In *International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 73–77. IEEE, 2011.

[114] Lijun Tang and John R Kender. Educational video understanding: Mapping handwritten text to textbook chapters. In *International Conference on Document Analysis and Recognition*, pages 919–923. IEEE, 2005.

[115] Lijun Tang and John R Kender. A unified text extraction method for instructional videos. In *IEEE International Conference on Image Processing 2005*, volume 3, pages III–1216. IEEE, 2005.

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

[117] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *International Conference on Computer Vision*, pages 839–846. IEEE, 1998.

[118] Alejandro H Toselli, Moisés Pastor, and Enrique Vidal. On-line handwriting recognition system for Tamil handwritten characters. In *Pattern Recognition and Image Analysis*, pages 370–377. Springer, 2007.

[119] Øivind Due Trier, Anil K Jain, and Torfinn Taxt. Feature extraction methods for character recognition - A survey. *Pattern Recognition*, 29(4):641–662, 1996.

[120] Ba Tu Truong and Svetha Venkatesh. Video abstraction: A systematic review and classification. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 3(1):3, 2007.

[121] Howard Turtle and James Flood. Query evaluation: strategies and optimizations. *Inf. Process. Manage.*, 31(6):831–850, 1995.

[122] Szilárd Vajda, Leonard Rothacker, and Gernot A Fink. A method for camera-based interactive whiteboard reading. In *International Workshop on Camera-Based Document Analysis and Recognition*, pages 112–125. Springer, 2011.

[123] Cornelius J. van Rijsbergen. *Information Retrieval.* Butterworths, 2nd edition, 1979.

[124] Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *SIGIR*, pages 105–114, 2011.

[125] Yuehan Wang, Liangcai Gao, Simeng Wang, Zhi Tang, Xiaozhong Liu, and Ke Yuan. WikiMirs 3.0: A Hybrid MIR system based on the context, structure and importance of formulae in a document. In *Proceedings of the 15th ACM/IEEE-CE on Joint Conference on Digital Libraries*, pages 173–182. ACM, 2015.

[126] Markus Wienecke, Gernot A Fink, and Gerhard Sagerer. Toward automatic video-based whiteboard reading. *International Journal of Document Analysis and Recognition (IJDAR)*, 7(2-3):188–200, 2005.

[127] Peter Willett. Document retrieval experiments using vocabularies of varying size. II. Hashing, truncation, digram and trigram encoding of index terms. *J. Documentation*, 35:296–305, 1979.

[128] Ting-Fan Wu, Chih-Jen Lin, and Ruby C Weng. Probability estimates for multi-class classification by pairwise coupling. *The Journal of Machine Learning Research*, 5:975–1005, 2004.

[129] Kuldeep Yadav, Ankit Gandhi, Arijit Biswas, Kundan Shrivastava, Saurabh Srivastava, and Om Deshmukh. Vizig: Anchor points based non-linear navigation and summarization in educational videos. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pages 407–418. ACM, 2016.

[130] Mingqiang Yang, Kidiyo Kpalma, Joseph Ronsin, et al. A survey of shape feature extraction techniques. *Pattern Recognition*, pages 43–90, 2008.

[131] Qixiang Ye and David Doermann. Text detection and recognition in imagery: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7):1480–1500, 2015.

[132] Xu-Cheng Yin, Ze-Yu Zuo, Shu Tian, and Cheng-Lin Liu. Text detection, tracking and recognition in video: A comprehensive survey. *IEEE Transactions on Image Processing*, 25(6):2752–2773, 2016.

[133] Abdou Youssef. Roles of math search in mathematics. In *Mathematical Knowledge Management*, pages 2–16. Springer, 2006.

[134] Abdou S Youssef. Methods of relevance ranking and hit-content generation in math search. In *Towards Mechanized Mathematical Assistants*, pages 393–406. Springer, 2007.

[135] Ke Yuan, Liangcai Gao, Yuehan Wang, Xiaohan Yi, and Zhi Tang. A mathematical information retrieval system based on RankBoost. In *Proc. JCDL*, pages 259–260. IEEE, 2016.

[136] Richard Zanibbi, Akiko Aizawa, Michael Kohlhase, Iadh Ounis, Goran Topić, and Kenny Davila. NTCIR-12 MathIR Task Overview. In *Proc. NTCIR-12*, pages 299–308, 2016.

[137] Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. *IJDAR*, 15(4):331–357, 2012.

[138] Richard Zanibbi, Kenny Davila, Andrew Kane, and Frank Tompa. Multi-stage math formula search: Using appearance-based similarity metrics at scale. *SIGIR*, 2016.

[139] Richard Zanibbi and Awelemdy Orakwue. Math search for the masses: Multimodal search interfaces and appearance-based retrieval. In *Intelligent Computer Mathematics*, pages 18–36. Springer, 2015.

[140] Richard Zanibbi and Li Yu. Math spotting: Retrieving math in technical documents using handwritten query images. In *ICDAR*, pages 446–451. IEEE, 2011.

[141] Richard Zanibbi and Bo Yuan. Keyword and image-based retrieval of mathematical expressions. In *IS&T/SPIE Electronic Imaging*, pages 78740I–78740I. International Society for Optics and Photonics, 2011.

[142] Shiliang Zhang, Qi Tian, Qingming Huang, Wen Gao, and Yong Rui. Usb: ultrashort binary descriptor for fast visual matching and retrieval. *IEEE Transactions on Image Processing*, 23(8):3671–3683, 2014.

[143] Wei Zhang and Chong-Wah Ngo. Topological spatial verification for instance search. *IEEE Transactions on Multimedia*, 17(8):1236–1247, Aug 2015.

[144] Yimeng Zhang, Zhaoyin Jia, and Tsuhan Chen. Image retrieval with geometry-preserving visual phrases. In *CVPR*, pages 809–816. IEEE, 2011.

[145] Wei Zhong and Hui Fang. OPMES: A similarity search engine for mathematical content. In *ECIR*, pages 849–852. Springer, 2016.