Issues in Performance Evaluation of Mathematical Notation Recognition Systems

by

Adrien Lapointe

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
May 2008

Abstract

Performance evaluation of document recognition systems is a difficult and practically-important problem. In this thesis, we contribute to the understanding of performance evaluation by studying some issues that arise in evaluation of systems for recognition of mathematical expressions. Issues that are discussed cover the reported performance evaluation experiments, the code availability, the nature of the mathematical notation, the extent of the coverage of mathematical recognition systems, and the quantification of performance evaluation results. For each issue, we discuss its impact on performance evaluation, give an overview of the state of the art for addressing it and point out open problems.

Acknowledgments

La rédaction d'un document de l'ampleur d'un mémoire de maîtrise demande énormément de travail. Je tiens à exprimer ma gratitude à tous ceux qui ont contribué au succès de cette entreprise. Tout d'abord, j'aimerais remercier Dorothea Blostein qui fut une directrice de thèse attentive et dévouée. Elle n'a jamais ménagé ses efforts pour m'aider à améliorer ce mémoire. Je tiens aussi à remercier Sonya qui m'a toujours soutenu, qui a su me motiver lorsque je doutais et qui est une épouse formidable. Merci aussi à ma famille et ma belle-famille pour leurs encouragements. Un remerciement tout spécial à mes amis David van Geyn, James Dutrisac et Guillaume Grenier-Lachance pour leur soutien moral. Un grand merci à Richard Zanibbi pour ses conseils et pour FFES/DRACULAE qui fut très utile au développement de ma thèse. Merci à George Labahn et son équipe de recherche de m'avoir permis d'utiliser MathBrush. Enfin, je veux remercier toute l'équipe de soutien du département de science informatique : Debby Roberston, Tom Bradshaw, Gary Powley et tous les autres qui contribuent au bon déroulement des programmes de 2^{e} et 3^{e} cycle.

Contents

List of Tables

List of Figures

Glossary

Bitmap A data structure that represents images as a rectangular grid of pixels.

BMP A file format for bitmap images. The filenames extension is `.bmp`.

FFES/DRACULAE A Mathematical recognition system that is composed of a handwritten mathematical character recognizer, FFES and a parser, DRACULAE. FFES was developed by Steve Smithies, Kevin Novins and Jim Arvo [35]. DRACULAE was developed by Richard Zanibbi [44].

GIF A bitmap image format that stands for Graphics Interchange Format. It was introduced by CompuServe. The filename extension is `.gif`.

Ground Truth The expected output for a given input in a dataset. For instance, the LaTeX ground truth of the mathematical expression $\frac{a^2+b}{c}$ is "`\frac{a^2+b}{c}`".

HMM Hidden Markov Model (HMM) is a statistical model in which hidden parameters are determined from observable parameters. The model is then used to perform further analysis.

InftyReader A Mathematical recognition system that can handle full document page and give output in many formats. The system was developed by members of the Infty Project [2].

JPEG An image compression method and format. The name comes from the Joint Photographic Experts Group that defined the method. The filenames extensions are `.jpg` and `.jpeg`.

LaTeX A document preparation system and markup language. It has a mode that allows to visually represent mathematical notation.

Maple A commercial software for computer algebra that was developed by the Symbolic Computation Group at the University of Waterloo. It features code that allows representing mathematical expressions.

Mathbrush A Mathematical recognition system that can handle handwritten mathematical notation. It is currently under development by the Symbolic Computation Group at the University of Waterloo.

Mathematica A commercial software for computer algebra that features a language for representing mathematical expressions.

MathML The Mathematical Markup Language (MathML) is an application of XML that can describe both structure and content of mathematical expressions.

Maxima A free computer algebra system.

OCR Optical Character Recognition (OCR) is a translation of typeset or handwritten text to machine-editable text.

PDF The Portable Document Format is a fixed-layout format used for representing documents. It was developed by Adobe.

XML The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages.

Chapter 1

Introduction

Performance evaluation has been a major problem in computer science for decades. In 1967, Calingaert already notes the increasing difficulty in evaluating systems as they are getting more complex [11]. Not only has performance evaluation been a problem for a long time, but it is an essential component in the development of computer systems since it can give useful information to users and developers of those systems. Performance evaluation provides developers with a way of measuring whether progress is actually taking place, and help in planning the next implementation step [18, 25]. At the end of the development process, performance evaluation can be a tool for quality assurance and behaviour prediction and a basis for acceptance testing [11, 18]. Moreover, it can be used in some cases for troubleshooting. Sometimes performance evaluation can indicate the reasons behind errors [18]; this can help to further improve systems [40, 43]. For instance, Chan and Yeung and Takiguchi *et al.* use evaluation to assess the efficiency of the error correction mechanism of their respective mathematical recognition systems and to make improvements as needed [13, 39]. Performance evaluation can also provide a solid basis for making decisions, such as deciding how to

upgrade existing systems, deciding what method is best suited for a given application, or improving algorithm design [11, 42, 43]. Lastly, performance evaluation can be used to establish a solid knowledge of the state of the art in a given research area [42, 43].

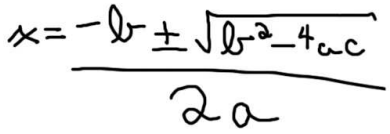
According to the Oxford English Dictionary, *performance* is "the quality of execution of an action, operation, or process" and *evaluation* is "The action of evaluating or determining the value of" something [1]. So performance evaluation in computer science is the action of determining the quality of execution of computer systems. However, this general definition is adapted differently to various fields of computer science. For instance, in natural language processing (NLP), performance evaluation is defined by Guida and Mauri as the task of comparing the actual behaviour of a system with the original user requirements and of measuring the possible deviations [18]. In graphics recognition, Wenyin and Dori define performance evaluation as the quantification of the difference between the actual output of the system and the expected one that is found in the ground truth of the test set [43]. In this thesis, we are interested in the performance evaluation of document image analysis systems and we will adopt Wenyin and Dori's performance evaluation definition.

1.1 Mathematical Notation Recognition Systems

Document image analysis systems aim at converting document images to symbolic form for the modification, storage, retrieval, reuse, and transmission of documents [29]. In order to focus our investigation, we concentrate on systems for the recognition of mathematical notation.

Mathematical notation is in widespread use in all fields of science. The notation has been developed over centuries and is either typeset or handwritten (see

Figure 1.1).



(a) Handwritten Mathematics

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

(b) Typeset Mathematics

Figure 1.1: Graphical Representations of mathematical expressions.

With the widespread use of computers, there is a need to be able to use mathematical notation in electronic formats. Consequently, many techniques for inputting mathematical notation in computers have been developed [9, 12]. The notation can be input using a manual technique such as a special keyboard, or a template and palette system, or key combinations, or typed input language such as LaTeX [12, 48]. The other possibility for inputting mathematical notation in computers is to use mathematical recognition systems, either with a scanner or a data tablet.

Mathematical notation recognition is a subfield of document image analysis that aims at translating mathematical notation into its corresponding computer form. While good progress has been made, there are limitations in present-days computer software for recognizing mathematical notation in the way it is used by humans.

Existing mathematical recognition software is not capable of error free understanding of typeset or handwritten mathematical expressions. As a result, it is common for users to proofread the systems' output in order to correct these errors. However, if too many errors are present, correction of the errors might be more time consuming then using a manual entry method. Consequently, the quantification of correctness is central when using mathematical recognition systems. Quantification helps users

to judge whether a given system is worth using. It also allows developers to know whether a system requires more development. Quantification is done using metrics. Some common metrics are *symbol recognition rate*, *error rate*, and *recall*. These are discussed further in Chapter 6.

1.2 Thesis and Contributions

Much work has been done on performance evaluation in the field of document image analysis [15, 18, 21, 25, 26, 32, 33, 40, 42, 43]. Some papers focus on the evaluation of mathematical recognition systems [7, 31]. Different issues are discussed in the literature, but no single place lists all the issues. Our major contribution to the field of performance evaluation in this thesis is a list of issues that arise when trying to evaluate the performance of mathematical recognition systems.

Issue 1: System performance is not always fully reported in the literature.

Issue 1.1: Reported performance measures are hard to compare between systems.

Issue 2: Performance evaluation techniques that are used in the literature are not always reported in enough detail to allow reproduction of the experiment.

Issue 3: Third-party performance evaluation depends on the availability of the source code and the executable code.

Issue 4: The levels of mathematical notation are central to the evaluation process, but they are poorly defined.

Issue 4.1: Performance evaluation must take into account that mathematical expressions are represented at various levels.

Issue 5: Performance evaluation must take into account the various equivalent representations within the computer mathematical formats.

Issue 6: Performance evaluation methods should reflect the systems coverage.

Issue 7: It is difficult to determine the extent of coverage of a mathematical recognition system.

Issue 8: Standardized datasets are needed for effective performance evaluation.

Issue 8.1: Datasets should be representative of a known domain, but domains are difficult to characterize and the representativeness of a dataset is difficult to assess.

Issue 9: Performance evaluation often involves matching output to ground truth. Appropriate matching techniques are difficult to define.

Issue 10: It is difficult to evaluate the quality of performance metrics and to select the best metrics for quantifying system performance.

Issue 10.1: Different metrics reflect various aspects of performance. It is not clear how best to combine metrics.

Issue 11: Performance metrics cannot always be applied directly to the output produced by a recognition system. Sometimes the metrics need to be modified to adapt to the type of output.

Issue 12: Performance evaluation is affected by the setting of system parameters. It is difficult to determine if parameter settings are optimal or not.

These issues are defined and discussed in Chapters 2 to 6. We discuss the impact of each issue on performance; by giving an overview of the state of the art for addressing each issue; and by pointing out open problems.

The thesis is organised as follows. In Chapter 2, we examine the difficulty of evaluating systems based on their reported performance measures and the problems in reproducing reported evaluation techniques. In Chapter 3, we cover the impact of the availability of an executable version of the system and/or its source code on the selection of an evaluation method. In Chapter 4, we look at issues that are related to two inherent characteristics of mathematical notation: levels of mathematical notation and equivalent representations within computer mathematical formats. We also

look at the consequences these two characteristics have on the performance evaluation of mathematical recognition systems. In Chapter 5, we discuss issues that come from the difference in coverage on performance evaluation and present some techniques to determine system's coverage. Lastly, in Chapter 6, we examine issues that are related to the quantification of performance evaluation results. More precisely, we look at issues related to standard datasets; the difficulty of defining appropriate matching techniques for ground truth and output; issues related to the quantification of performance using metrics; and, the importance of algorithm parameters in the evaluation of performance. The presentation of these issues provides a clear overview of the many challenges and open problems in performance evaluation.

Chapter 2

Issues Related to Reported Performance Evaluation Experiments

In this chapter, we examine the difficulty of evaluating systems based on reported performance measures and the problems in reproducing reported evaluation techniques. In Section 2.1, we look at the difficulties in evaluation that arise from the way in which performance measures are reported in the literature. In Section 2.2, we look at the impact of the way in which evaluation techniques are reported on the reproduction of performance evaluation experiments.

2.1 Comparison of Different Systems Based on Reported Performances

Reported performance measures provide a direct way to assess the performance of a system.

Issue 1: *System performance is not always fully reported in the literature.*

Miller and Viola for example report results from only a few tested expressions to compare two different algorithms for pruning the parsing search space [28]. They show examples of success and failure of the recognition process, but do not provide quantified performance measure. As another example, Lavirotte and Pottier do not report any evaluation of the system they describe, although they imply that some testing has been done [24]. Moreover, even when performance measures are reported, they are not necessarily easy to use.

Issue 1.1: *Reported performance measures are hard to compare between systems.*

One of the reasons why reported performance measures are hard to compare is that different authors use different metrics and datasets to report the performance of their systems [45]. Table 2.1 on page 12 shows an overview of some metrics and datasets reported in the literature. This is not an exhaustive list since we will discuss more about datasets and metrics in chapter 6. Nevertheless, it illustrates the variety of datasets and metrics that are used in the literature to report system's performance. One can easily see that every author uses a different dataset and that no common dataset stands out. Similarly, there is a variety of metrics, and some of them, such

as *symbol recognition rate* and *expression recognition rate*, are used by more than one author. Unfortunately, even if they are used by more than one author, these two metrics do not provide a comprehensive way to compare system's performance. *Symbol recognition rate* only covers one aspect of mathematical recognition systems: symbol recognition. *Expression recognition rate* does not give any partial credit, which results in treating an expression that contain only one recognition error the same as an expression that contains many recognition errors.

Another reason for the difficulty in comparing reported performance is that their statistical significance varies. Some authors use small datasets, while others use larger datasets. Due to the lack of standard datasets, some authors use only a few examples to measure the performance on. For instance, Takiguchi *et al.* asses the character recognition module using 30 expressions and the structural analysis module using three expressions [38]. The effect of reporting performance on a limited sample is that the result may not be representative of the performance on another test set.

In conclusion, mathematical recognition systems cannot always be assessed based on their reported performance measures. Even when reported performances can be used, they can be misleading and additional testing may be required in order to get an idea of systems performance or to validate reported measures.

2.2 Reproducibility of Performance Evaluation Experiments

A key feature of the scientific method is the ability to reproduce an experiment as a way to validate or negate the results.

Issue 2: *Performance evaluation techniques that are used in the literature are not always reported in enough detail to allow reproduction of the experiment.*

Many aspects of performance evaluation can be affected by the lack of details in the reported evaluation technique. Insufficient detail sometimes occurs with the definition of metrics. For example, Garain and Chaudhuri define *levels* [1] as an indication of the complexity of a mathematical expression which they use in the definition of their performance evaluation metric [17]. According to their definition, *levels* are horizontal lines centered on symbol centroids. Centroids that are nearly collinear are grouped on the same *level* line. Figure 2.1 shows an example of *levels* according to their definition.

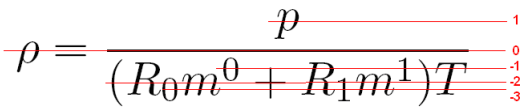


Figure 2.1: Example of levels as reported in [17]

A serious problem with this definition of *levels* is that the authors give no indication on how far centroids have to be from each other in order to warrant creation of another *level* line. With such ambiguities, more than one set of *levels* can apply for some expressions. For example, in figure 2.1, if the centroids of the two exponents of the m's in the denominator were at slightly different vertical positions, there might be two *levels* instead of the single level marked -1. Clarification of the definition of *levels* is critical in this case because the authors base the definition of their metrics on it. These metrics are then used to report the performance of their system. Reproducing the authors' experimental results thus become dependent on our assumptions.

[1] Here, level has another meaning than the one we will use in the rest of the thesis.

In some cases, insufficient detail is given about the experimental conditions. For example, Ashida *et al.* report symbol recognition rate in two different scenarios [7]. In the first one, a distinction between similar symbols is made and in the second one no distinction is made. However, they only give few examples of symbols considered as similar ("0", "O", "o"; "1", "l"; "S", "s"), and consequently it is not clear exactly which are considered *similar*. This is a major problem for reproducing the results. It is also a problem when we want to compare these results with others. For instance, consider another paper which reports recognition rate in a similar way, but does not consider "S" and "s" as similar symbols.

These examples illustrate two cases where there is a lack of details. In all cases, lack of detail forces assumptions to be made in reproducing evaluation experiments. A possible way to avoid this problem is to follow guidelines such as the one suggested by Haralick [19]. He notes three components that should be included in controlled experiments. These components can be a useful indication of what should be included to make a reported evaluation unambiguous. The first one is a measurement plan that indicates what will be measured, how it will be measured and with what accuracy. The second one is the sampling design that describes the dataset and how it is sampled for performing tests. Finally, the third one is a statistical analysis describing how the raw data will be analyzed.

Authors	Metrics	Dataset
Ashida *et al.* [7]	• Symbol recognition rate	2,100 pages from *Archiv der Mathematik* and *Commentarii Mathematici Helvetici* with ground truth from the authors.
Chan and Yeung [13]	• Expression Recognition Rate. • Symbol recognition rate. • Operator recognition rate. • Integrated Performance Measure.	600 expressions from *CRC Standard Mathematical Tables and Formulae* [49] written by ten people.
Garain and Chaudhuri [17]	• Global performance index • Average performance index	400 pages from various sources with ground truth from the authors
Kosmala *et al.* [23]	• Computing time	One expression with 116 symbols.
Okamoto *et al.* [31]	• Recognition Rate	96 pages from *Archiv der Mathematik* with ground truth from the authors.
Takiguchi *et al.* [39]	• Recognition Rate	15 expressions from *Fundamental Formulas of Physics* [27] with no specifications on the ground truth.
Zanibbi *et al.* [45]	• Baseline recognition rate • Token placement rate • Expression recognition rate	University of Washington database III [30]

Table 2.1: List of metrics and datasets and their references.

Chapter 3

Issues Related to Code Availability

In this chapter, we cover the impact of the availability of an executable version of the system and/or its source code on the selection of an evaluation method. In Section 3.1, we examine the different scenarios of code and system availability and in Section 3.2 we look at approaches to performance evaluation that are adapted to the different availability scenarios. Finally, in Section 3.3, we discuss different factors that affect the selection of the approach.

3.1 Code Availability

In addition to using reported performance measures or reproducing evaluation experiments (see Chapter 2), evaluators can also carry out their own testing on existing systems.

Issue 3: *Third-party performance evaluation depends on the availability of the source code and the executable code.*

There are three cases of code and system availability that evaluators must deal with:

1 **Source Code Available :** Some systems are fully open source and available over the Internet. This is the case with FFES/DRACULAE [4].

2 **Executable Code Available :** In some other cases an executable version of the system may be available by request to the authors or through the Internet. For instance, InftyReader, an OCR software that can handle mathematical expressions, is offered through the authors' website as a 30-day trial version and can be purchased for further use [2]. MathBrush, another mathematical recognition system, developed at the University of Waterloo, is not distributed freely over the Internet, but we have obtained a copy through a request to the authors [3]. In both cases, the source code is not available and only a running version of the system is available.

3 **Code Unavailable :** In many cases, systems are informally described in literature and neither their source code nor a compiled version of them is available [13, 14, 24, 38, 39].

Since code is not always available, the evaluation techniques must be adapted.

3.2 Adapted Evaluation Approaches

We describe two approaches to performance evaluation that are adapted to the availability scenarios: white box evaluation that can be used when the source code is available and black box evaluation that can be used when only executable code is

available. For the cases where neither source code nor executable code is available, we can re-implement the system. Re-implementation is not in itself an approach to performance evaluation, but it can permit the implementation of either a white box or a black box evaluation.

3.2.1 Black Box Evaluation

Black box evaluation consists of taking the output of a system for a given input and comparing it with the ground truth. This evaluation technique evaluates the product of a system instead, but not the process that caused it [18, 40].

It is possible to use black box evaluation in cases where the source code is available as well as in cases where only an executable version of the system is available. It is by far the most common evaluation method, and most performance measures reported in the literature are the result of black box evaluations.

The major shortcoming of this approach, from a developer perspective, is its lack of insights on the internal functions of a system which can make error diagnosis and system improvements harder. A possible solution to this problem is to use white box evaluation.

3.2.2 White Box Evaluation

The second approach to performance evaluation, white box evaluation, can be done in various ways. It is usually performed on an ad-hoc basis where users add code to produce additional outputs reflecting partial results and possibly indicating where errors were made [47]. Zanibbi *et al.* propose another approach that uses decision based method where a hypothesis history is recorded [46, 47]. Regardless of how it

is performed, white box evaluation sheds light on the internal process that produces an output from a given input. This evaluation approach can be used in cases where evaluators have full access to source code. White box evaluation is not as common as black box evaluation because it is typically harder to perform.

One of the major problems with this approach to performance evaluation is that implementations are quite different from one system to the next. Not only are numerous programming languages used, but different techniques have intrinsic fundamental differences. For instance, Blostein and Grbavec note five different categories for symbol arrangement analysis: syntactic method, projection profile cutting, graph rewriting, procedurally-coded math syntax and data-driven/knowledge-driven modules [9]. The internal data in each category of implementation is different and comparing partial results is challenging. Zanibbi *et al.* propose a solution to this problem [46, 47]. They suggest re-implementing parts of the system using a programming language that makes recognition decisions explicit, and then comparing internal mechanisms based on decisions.

3.2.3 Re-implementation

Re-implementation consists of creating code from the reported description of an algorithm. It puts the evaluator in a situation where he/she has access to the source code and consequently to either black box or white box evaluation approaches.

Re-implementation can be used when neither an executable version of the mathematical recognition system nor the source code is available. It can also be used to allow white box evaluation in cases where only an executable version of the system

is available or in cases where the source code is available to make white box evaluation easier [47]. For instance, re-implementation is used by Zanibbi *et al.* to allow evaluation of two table recognition methods that are described in the literature [47].

There are some shortcomings related to the re-implementation strategy. Firstly, the quality and amount of detail present in the published description will sometimes only allow partial re-implementation of a method. The second shortcoming is that even when full re-implementation is possible, chances are that the system will not be identical to the original one and thus the evaluation does not fully reflect the original system's performance. Other shortcomings are that re-implementation is time consuming and error prone. Moreover, re-implementation is not possible when published descriptions are incomplete.

3.3 Selection of Adapted Evaluation Approach

The selection of the appropriate evaluation approach is most of the time dictated by the availability of executable version and source code. However, there are also other factors that determine the choice.

One of these factors is the type of evaluator. An evaluator that is a developer will have different evaluation goals than one that is a user. Developers will most likely want evaluation to support development by indicating the system's weaknesses and where development efforts are needed. On the user side, the evaluation goal will most likely be to allow a more enlightened choice of system. Developers will want to have insights on what is going on internally whereas users will not necessarily want as much detail. As a result, white box evaluation might be more appropriate for developers and black box evaluation for users.

It is also important to note that the distinction between black box and white box evaluation is not always clear. Sometimes authors report on evaluations that are primarily black box, but contain some amount of white box evaluation. For instance, it is quite common that authors will assess the performance of symbol recognition and structure analysis separately [7, 13, 17]. This gives some insights on the internal process, which is like white box evaluation, but each module is evaluated by looking at its input and output, which is like black box evaluation.

Chapter 4

Evaluation Issues Related to the Nature of Mathematical Notation

This chapter covers issues that are related to two inherent characteristics of mathematical notation and the consequences they have on the performance evaluation of mathematical recognition systems. In Section 4.1, we discuss the intrinsic levels of mathematical notation and the influence they have on performance evaluation. In Section 4.2, we discuss the consequences of the many equivalent representations in the computer mathematical formats.

4.1 Levels of Mathematical Notation

Mathematical notation can be described using different levels of representation related to the appearance and meaning of the notation.

Issue 4: *The levels of mathematical notation are central to the evaluation process, but they are poorly defined.*

The precise definition of levels could be by itself the subject of research. As an approach to defining levels, we use three different strategies for informally detemining a set of levels that is useful for evaluation of mathematical recognition systems.

Firstly, we can form groups of formats that are used to represent mathematical notation in computers. The different computer mathematical notation can be grouped into levels, where the formats within a level encode information in a similar way, but the formats in different levels encode the information in different ways. For instance, bitmap files and PDFs encode mathematical expressions as arrays of pixels or as vectors with no insight on the meaning. On the other hand, LaTeX files explicitly identify symbols and structures. Using this grouping strategy we can define three levels: *document-image level* for formats such as PDF, BMP, JPG, GIF; *structure level* for formats such as LaTeX and Presentation MathML; and *computer algebra level* for formats such as Maple, Maxima, Mathematica, and Content MathML

Secondly, we can study the formats that mathematical recognition systems use as input and output. Mathematical recognition systems use different formats for their input and output. For instance, Infty Reader takes a document image as input (e.g. PDF, JPG, BMP) and produces output in a variety of formats (LaTeX, MathML, XML) [2]. On the other hand, the parser module of FFES/DRACULAE, DRACULAE, takes an attributed list of symbols with bounding box as input and produces output in LaTeX and in an operator tree format [4]. The symbols in the attributed list are usually referred to as segmented symbols. Figure 4.1 shows an example of an expression where the symbols are segmented. As another example, MathBrush takes a vector image as input and produces an output in various formats including Maple [3]. By studying the input and output formats of these three systems, we can define

four levels: *document-image level* which accounts for the input format of Infty Reader and of MathBrush; *segmented symbols level* which accounts for the input of DRACULAE; *structure level* which accounts for the output formats of Infty Reader and for the LaTeX output of DRACULAE; and the *computer algebra level* which accounts for the output format of MathBrush and the operator tree format of DRACULAE.

(a) Segmented Expression

```
y <(35,24),(71,69)>
- <(32,69),(77,79)>
2 <(39,89),(67,134)>
```

(b) Segmented Expression

Figure 4.1: An example of segmented expression where the segmented symbols are enclosed in a bounding box. (a) Visual representation of the expression and the bounding boxes. (b) A possible text representation of the segmented expression. The symbol's identities are followed by the coordinates of the upper-left and lower-right corners of their bounding boxes.

Thirdly, we can look at the various steps used by mathematical recognition systems to perform recognition. Different mathematical recognition systems use different steps to process the input. The formats that are used as input and output of the various steps can help define levels. In their survey of mathematical recognition systems, Chan and Yeung note four processing tasks that are used in the different systems: symbol segmentation, symbol recognition, identification of implicit operators, and structural analysis [12]. Looking at these processing steps, we can define the *document-image level*, the *segmented symbols level*, the *structure level* and the *computer algebra level*.

In combining the results from these three strategies, we identify four levels that are used in this thesis: the document- image level, the segmented symbols level, the

structure level and the symbolic algebra level. Figure 4.2 displays the four defined levels along with an example for each one of them. The concept of levels accounts for the differences and similarities between computer mathematical formats, as well as the passage from one level to another through the various recognition steps. Formats that encode information in a similar fashion should be at the same level, whereas those that encode it differently should be at two distinct levels.

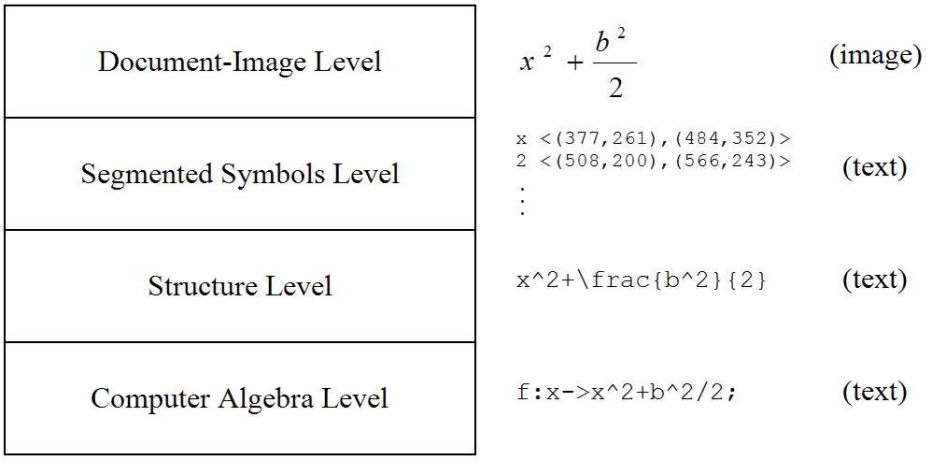


Figure 4.2: Four levels of representation for mathematical notations.

Unfortunately, defining the levels precisely is problematic for many reasons. Firstly, the difference between various formats is often blurred. For instance, a document that is in the LaTeX format can also include images. Secondly, the order of the recognition steps varies. Some systems perform symbol recognition and then structural analysis whereas others perform structural analysis prior to symbol recognition [12]. Lastly, the distinction between the steps is not always clear. For instance, Miller and Viola and Suzuki *et al.* describe systems where many candidates are maintained after the character recognition step and the final result only decided during the structural analysis step [28, 36].

These four levels are defined broadly, and do not take into account all the possible recognition steps and formats. Nevertheless, these levels provide a basis for describing the general order of recognition steps, which goes from the document image level to structure, and possibly on to the symbolic algebra level.

The existence of levels has an impact on performance evaluation.

> **Issue 4.1:** *Performance evaluation must take into account that mathematical expressions are represented at various levels.*

The first aspect of the levels that must be considered during evaluation is levels of the input and output of the mathematical recognition systems. Systems that produce output at different levels cannot be compared directly. For instance, a system that goes from the document-image level or the segmented symbols level to the computer algebra level needs more external information than one that goes from the document-image level to the structure level. A possible way to compare systems that span different levels is to isolate parts of the system. For example, if we want to compare a system that takes input at the document-image level with one that takes input at the segmented symbols level, we can give both systems input at the segmented symbols level, thus bypassing the symbol-recognition portion of the first system.

The second aspect of the levels that must be considered during evaluation is the matching of the output and ground truth. The dataset needs to take into account the different levels and the formats that come with them. Ideally, a dataset includes inputs and ground truth at different levels. So for instance if we are testing a mathematical recognition system that goes from the document-image level to the structure level, we would have inputs that are at the document-image level and ground-truth at the structure level. The dataset from the University of Washington (see section 6.1 for

more details), for example, includes expressions as bitmaps, as an attributed list of symbols with bounding boxes coordinates, as well as their representation in the LaTeX format.

4.2 Equivalent Representations of Mathematical Notation

There are equivalent ways to represent expressions within the different computer mathematical formats.

Issue 5: *Performance evaluation must take into account the various equivalent representations within the computer mathematical formats.*

This is a major issue when comparing a system's output with the ground truth since all the equivalent representations have to be treated as equal. The output of the system and the ground truth can be equivalent but different. For example, the expression with image x_0^2 can be represented by several equivalent LaTeX strings, including "`x^2_0`", "`x_0^2`", "`x^{2}_{0}`", and "`x_{0}^{2}`".

We experienced the problem of many equivalent representations for the same expression when testing the mathematical recognition system FFES/DRACULAE on the mathematics pages from the University of Washington database (UW-III) [4, 30]. We noticed that FFES/DRACULAE always outputs expressions with subscript and superscript arguments fully parenthesized and that the subscript always comes first and the superscript always follows, when both are present. For example, x_0^{i+1} is outputted by FFES/DRACULAE as "`x_{0}^{i+1}`". On the other hand, we noticed that the LaTeX ground truth of UW-III is not as consistent. In some cases where there

is only one argument to the subscript or the superscript, the brackets are omitted, which is legal in LaTeX. So the previous example, x_0^{i+1}, is written as "`x_0^{i+1}`" in the ground truth. We also noticed that the order of the subscripts and superscripts, when both are present, is not always the same; in some ground truth files, the subscript is first whereas in others it is second. Thus, matching the output of DRACULAE with the LaTeX ground truth of UW-III is challenging because all the different equivalent representations need to be accounted for.

The subscript and superscript example demonstrates one case of equivalence in LaTeX. There are many other such cases in LaTeX, and similar cases occur in other computer mathematical formats. For instance, Archambault and Moço note that there are many equivalent ways to represent the same expression in MathML [6].

`a^2_0`	is not in canonical form
`a_0^2`	is not in canonical form
`a^{2}_{0}`	is not in canonical form
`a_{0}^{2}`	is in canonical form

Table 4.1: Four LaTeX strings representing the expression a_0^2. Only the fourth string is in the canonical form we defined for our evaluation of the mathematical recognition system FFES/DRACULAE [4].

A *canonical form* can provide a possible solution to the problem of many equivalent representations for the same expression [5, 6]. A *canonical form* is a standard that provides a unique way to represent a given expression within a computer mathematical format. To solve the subscript/superscript problem that we encountered while evaluating FFES/DRACULAE, we defined a canonical form where every subscript and superscript needs to be parenthesized and the subscript, if present, always

comes before the superscript. This is illustrated in Table 4.1 where the fourth LaTeX string is in canonical form and the others are not. Our canonical form is not comprehensive since it does not cover all equivalences that are possible in LaTeXẆe devised this limited canonical form only to solve a particular problem we encountered during experimentations. A more comprehensive example of canonical form is described by Archambault and Moço [6].

It must be noted that a canonical form does not solve every equivalence problem. For example, $\log_e x$ and $\ln x$, $a + b + c$ and $c + a + b$, $\frac{x}{y}$ and xy^{-1} are pairs of expressions that have an equivalent meaning, and when evaluating at the computer algebra level, we would probably want each pair to be considered equivalent. Unfortunately, defining a canonical form to cover all these cases is extremely difficult.

Chapter 5

Issues Related to the Coverage of Mathematical Recognition Systems

In this chapter, we discuss issues that come from the difference in coverage. In Section 5.1, we look at the consequences of coverage on performance evaluation and the importance of reporting coverage. In Section 5.2, we discuss the different techniques to estimate the coverage of mathematical recognition systems.

5.1 Coverage of Mathematical Recognition Systems

The domain of all possible expressions in mathematical notation is very large. As a result, existing mathematical recognition systems only cover a certain subset of this domain.

Issue 6: *Performance evaluation methods should reflect the system's coverage.*

If two systems have the same coverage and produce output at the same level (see issue 4.1), they can be compared directly. However, if two systems differ in coverage,

the performance measure must take this into account. It is of little use to report the performance of a system that has been tested on a set of symbols, structures or dialects that the system was not designed to recognize. For instance, consider two systems where the first one covers a given domain of the mathematical notation and the second one covers a larger one as shown in Figure 5.1. Testing the systems on only one of the two domains would result in lower performance measures for the system that is not tested on its handled domain. One possible solution to avoid such distortion of the performance evaluation results is to compare performance only on the intersection of coverage. So for the example in Figure 5.1, the evaluation would be done on the intersection of domain 1 and 2.

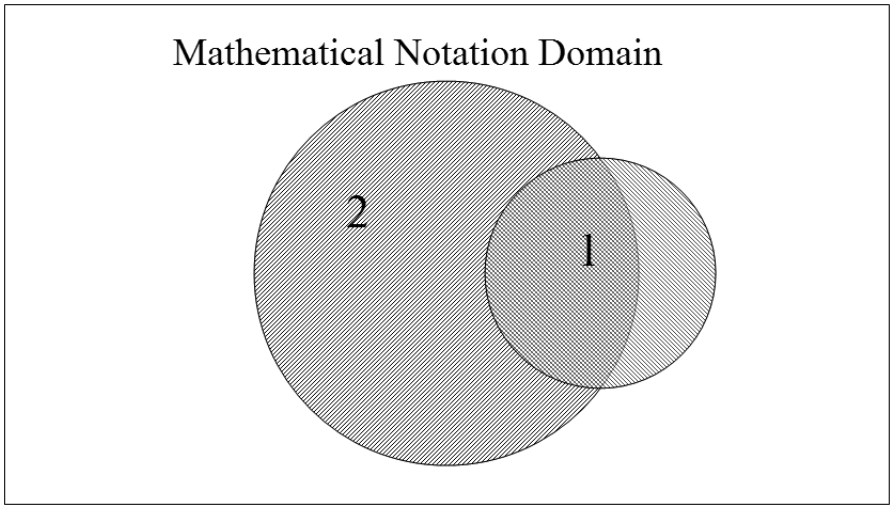


Figure 5.1: An example of mathematical notation domain coverage by two recognition systems.

Moreover, it is important to state the relative size of coverage when presenting the performance results. Some systems have high performance on a small domain

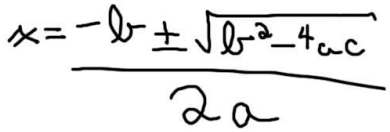
while others have lower performance, but on a larger domain. For example, testing systems 1 and 2 from Figure 5.1 on the intersection of their domains could tell us that system 1 has a better performance than 2. However, the extent of the domain is also an important aspect to consider. Systems with larger coverage are useful because of their increased generality and applicability.

One possible way to take into account the extent of the coverage when reporting performance is to make up a combined performance metric with weighted terms for accuracy and coverage. Unfortunately, as discussed in Section 6.3.2, combination of different performance metrics is hard and it is usually left informal, with separate values used to characterize accuracy and coverage.

Also, another problem that arises when reporting coverage is that it is not simple to quantify coverage. Ideally, we would like to be able to say that system x has recognition rate y on $z\%$ of the mathematical notation domain. Unfortunately, this solution is impractical because it is not possible to define the full domain of mathematical notation. Also, such a description of the coverage would not clearly state which symbols are covered.

Another aspect of coverage that must be considered is the type of mathematical notation that the recognition system handles. Mathematical notation can be handwritten or typeset (see Figure 5.2). Some systems are designed to handle handwritten mathematical notation whereas others are designed to handle typeset mathematical notation.

It is not appropriate to directly compare performance measures between systems that recognize handwritten notation and typeset notation. Handwritten mathematical notation is noisier than typeset mathematical notation. The symbols sometimes



$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

(a) Handwritten Mathematics (b) Typeset Mathematics

Figure 5.2: Graphical Representations of mathematical expressions. Same as figure 1.1, repeated here for convenience.

overlap which results in more segmentation errors (Figure 5.3). The symbols can be messy causing recognition errors. And, the symbol placement is variable resulting in structure analysis errors.

(a) (b)

Figure 5.3: An example of a handwritten expression. Segmentation (a) is correct, whereas (b) is incorrect.

A possible approach to compare systems that handle different types of mathematical notation is to bypass the symbol recognition step. This can be done by giving both systems input at the segmented symbols level. If we use segmented symbols that come from typeset expressions, both type of systems are comparable. However, if we use segmented symbols that come from handwritten expressions, chances are that the system that was designed for handwritten input will have better performance: that system can better cope with variability in layout of handwritten mathematical notation. Such variability is reflected in the bounding box coordinates of segmented

symbols from handwritten expressions since they were not necessarily following a constant baseline as it would happen in typeset mathematics.

5.2 Estimating Coverage

As discussed in the previous section, knowledge of the coverage of a mathematical recognition system is a key component to perform sensible evaluation and to report the performance.

Issue 7: *It is difficult to determine the extent of coverage of a mathematical recognition system.*

We have identified four different techniques for estimating the coverage of a system:

1 **From the literature:** We can estimate the coverage by looking for descriptions of the domain in the literature.

2 **From experiments:** We can estimate the coverage by testing different symbols, structures and function names on the system.

3 **From source code:** We can estimate the coverage by looking at the source code for indication of the coverage.

4 **From the training process:** We can estimate the coverage by examining all the options in the training process.

5.2.1 Estimating Coverage from Literature

It is sometimes possible to partly determine the coverage of a mathematical recognition system from descriptions of the domain found in the literature. Table 5.1 on

page 36 shows references to articles in which it was possible to find such descriptions along with the coverage we retrieved for the described systems.

In Table 5.1, one can see that the descriptions of the domain are not always clear and the coverage is not always fully and explicitly stated in the literature. For example, Takiguchi *et al.* provide an enumeration of some of the symbols, structures and function names that are handled by their system, but they do not fully describe what operator symbols and brackets are covered [38]. Similarly, Kosmala *et al.* list the symbols that are supported by their mathematical recognition system, but they do not mention which structures their system handles [23]. Ashida *et al.* say that their system handles Latin alphabet, Greek alphabet, digits and mathematical symbols, but they do not enumerate the mathematical symbols, only listing ( ), $\rightarrow$, $\cup$ [7]. They also say that function names are handled, but they only list *lim* and *sin*. As a result, the reader does not know precisely what the coverage of their system is.

5.2.2 Estimating Coverage from Experiments

Another approach to estimate the coverage of a system is to perform experiments on it. To experimentally estimate the coverage of a mathematical recognition system, we feed symbols, structures and function names to the system and look at the output to see if they have been recognized.

We have tried this approach on FFES/DRACULAE [4] to estimate which symbols and structures are covered[1]. For the experiment, we made a list of all the symbols and structures we wanted to test (see appendix A). We used a Wacom Tablet connected to

[1]We did not test the system for function names in that experiment because it would have required too many corrections of the character recognition output. We will show a better way to estimate which function names are handled later in this section.

a computer with Ubuntu 7.10 as the operating system. Using the FFES/DRACULAE 0.4 graphical user interface, we inputted each symbol and structure three times[2]. For the structures, we inputted simple examples such as $\frac{a}{b}$ to test the fraction structure. If the symbols or the structures were recognized at least once, we knew that they were covered. However, if the symbols or structures were not recognized, we were not able to determine whether it was handled.

Table 5.2 on page 37 shows the estimate of coverage we obtained through the experiment. By looking at this table, one can notice that only some letters from the Latin and Greek alphabets and some operator are included in the coverage. Some of the basic arithmetic symbols, such as $+$, $=$, $-$, are left out. In fact, one major limitation of this technique is that it is not always possible to deduce significant conclusions from unrecognized symbols or structures. Due to technical constraints, we only use three samples, and we cannot be certain whether or not an unrecognized symbol or structure would be recognized using further samples. Furthermore, the test is limited to a set domain of characters, and so it remains unknown whether the covered domain is larger or not. Regardless, this technique to delineate the coverage of a system is fairly simple to implement and can be applied to most mathematical notation recognition systems.

5.2.3 Estimating Coverage from Source Code

Another approach to estimate the coverage of a system is to look at the source code. We have tried this solution on FFES/DRACULAE and most specifically on the parser part, DRACULAE. We looked at the files `Expression.Grammar` and

[2]FFES was not trained to our specific handwriting, due to technical difficulties with the software.

`TeX\_ Symbol\_ List.Rules`. Table 5.3 on page 38 displays the coverage we obtained from this examination. One can see that this set of symbols and structures is much larger than the one we obtained through the experiment described in section 5.2.2. However, since we only looked at the parser's source code, DRACULAE, we cannot draw conclusions on the coverage of FFES/DRACULAE as a whole. It is quite possible that the character recognizer, FFES, cannot handle all of these symbols. Ideally, we would have looked at FFES's source code as well, but it is not as clear as DRACULAE's.

The difficulty of finding information about coverage within the code is a major problem with this technique. The information is not always explicit, and it is sometimes spread across different functions that are not always well commented. Another problem is that not all authors provide the source code (see section 3.1).

5.2.4 Estimating Coverage from the Training Process

Our last approach for estimating coverage is to go through the system's training process. This best applies to the symbol recognition of handwritten mathematical recognition systems where the symbol recognizer needs to be trained for the user's handwriting style. Typically, the user interface for training shows a symbol and asks the user to write it few times. The symbols that are shown are necessarily covered and thus we can recover part of the system's coverage.

We applied this technique to estimate the coverage of MathBrush [3]. The results are displayed in Table 5.4 on page 39. The estimated list of covered symbols is quite small, much smaller than sets of symbols estimated for FFES/DRACULAE (Tables 5.2 and 5.3). This illustrates a drawback of this technique: there is no way to

be sure that all the handled symbols are included in the training process. Moreover, Table 5.4 shows no information regarding the structure and function names that are covered. In fact, the structures and function names were not included in the training process of MathBrush and as a result it is not possible to know if they are covered or not. It is not clear whether or not other system's training processes include structures and function names. Lastly, this technique is not always available as not all systems have a training process. Regardless, this technique is useful since it can give a quick overview of the coverage.

Reference	Symbols	Structures	Functions
Takiguchi *et al.* [38]	Latin alphabet (Lower and Upper Case) Greek alphabet (Lower and Upper Case) Digits $+ \ - \ \times \ \div \ /$	Horizontal, fraction, subscript (right and left), superscripts (right and left), over, under, integral type, sum type	acos arccos arcsin arctan arg asin atan cosec cos cosec cosh cot coth csc deg det dim exp gcd hom inf ker lg lim liminf limsup ln log max min sec sin sinh sup tan tanh
Kosmala *et al.* [23]	Latin alphabet (Lower and Upper Case) Digits $+ \ - \ \cdot \ :$ $/ \ \overline{\quad} \ \hat{} \ \sqrt{\quad} \ \sum \ \prod$ $\int \ , \ ' \ = < > \leq \geq \rightarrow \leftrightarrow$ $\approx \ ! \ \infty$ $\alpha \ \beta \ \gamma \ \lambda \ \mu \ \Delta \ \pi \ \omega \ \epsilon \ \tau \ \phi$ $(\ ) \ [\ ] \ \{ \ \}$	Not Specified	Not Specified
Ashida *et al.* [7]	Latin alphabet, Greek alphabet, digits, double-struck ($\mathbb{A} \ \mathbb{J} \ \mathbb{P} \ \mathbb{Z}$), script ($\mathcal{A} \ \mathcal{E} \ \mathcal{F} \ \mathcal{Y}$), fraktur ($\mathfrak{A} \ \mathfrak{B} \ \mathfrak{F} \ \mathfrak{G}$), mathematical ($(\ ) \ \rightarrow \ \cup$)	Horizontal, fraction, subscript (right and left), superscripts (right and left), matrix, square root	lim sin
Okamoto *et al.* [31]	All symbols supported by LaTeX except the user defined ones.	Horizontal, fraction, subscript (right and left), superscripts (right and left), matrix, square root	Not specified

Table 5.1: Coverage of different systems described in the publications.

Type	**List**			
Latin alphabet	A	L	b	o
	C	M	g	p
	D	N	h	r
	H	S	i	w
	J	W	j	y
	K	Z	n	
Digits	$\{1, 2, 3, 4, 7\}$			
Greek alphabet	α	λ	ω	Ψ
	ε	μ	Γ	Ω
	ζ	ν	Λ	
	η	ρ	Π	
	θ	ψ	Σ	
Operators	$\sqrt{}$	!	$\int$	
Delimiters	(	)	[	]
	{	}		
Other	$\rightarrow$	$\perp$	∞	#
	$\Rightarrow$	$\cap$	?	
	$\exists$	$\therefore$	&	
	$\emptyset$	$\sim$	:	
Structures	Horizontal Fraction	Right Superscript Right Subscript	Sum Type	
Function Name	log			

Table 5.2: Coverage of FFES/DRACULAE [4] obtained from experiments.

Type	**List**					
Latin alphabet	A to Z a to z					
Digits	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$					
Greek alphabet	α	χ	ι	τ	Γ	Ψ
	β	ψ	κ	θ	Δ	Ω
	γ	δ	ν	υ	Λ	Υ
	η	λ	ξ	ϕ	Θ	
	μ	ϵ	π	ω	Ξ	
	ρ	ζ	σ	Π	Φ	
Operators	$=$	$\times$	$\leq$	$\equiv$	$\oplus$	$\sum$
	$\neq$	$\pm$	$\approx$	$\not\equiv$	$\otimes$	
	$+$	$>$	$\cong$	$\int$	$*$	
	$-$	$<$	$\simeq$	$\oint$	$\sqrt{}$	
	$\cdot$	$\geq$	$\sim$	∂	$!$	
Delimiters	$($	$)$	$[$	$]$	$\{$	$\}$
Other	∞	$\Uparrow$	$/$	$\supseteq$	$\bigcap$	$\angle$
	$\propto$	$\Leftrightarrow$	$\backslash$	$\notin$	$\bigcup$	$\models$
	$\emptyset$	$\therefore$	$:$	$\in$	$\exists$	$\nvDash$
	$\rightarrow$	$\&$	$\hat{}$	$\ni$	$\forall$	$\triangledown$
	$\leftarrow$	@	$\vec{}$	$\parallel$	$\neg$	$\square$
	$\leftrightarrow$	$\#$	$\subset$	∇	$\vee$	$\diamond$
	$\Leftarrow$	$\$$	$\not\subset$	$\Re$	$\wedge$	
	$\Rightarrow$	$\%$	$\subseteq$	$\aleph$	$\perp$	
	$\Downarrow$	$?$	$\not\subseteq$	$\acute{}$	$\top$	
Structures	Fraction	Horizontal				
Function Name	cos	tan	lg	min	exp	max
	sin	log	ln			

Table 5.3: Coverage of DRACULAE [4] obtained from examination of the code.

Type	**List**
Latin alphabet	A to Z a to z
Digits	$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Greek alphabet	α λ ξ Π β μ ζ Σ γ π Δ Ψ δ σ Γ ϵ θ Ω
Operators	$=$ $\neq$ $\geq$ $\surd$ $+$ $<$ $!$ $-$ $\leq$ $\equiv$ $\pm$ $>$ $\int$
Delimiters	$($ $)$ $[$ $]$ $\{$ $\}$
Other	∞ $\square$ $\rightarrow$

Table 5.4: Coverage of MathBrush [3] obtained from the training process.

Chapter 6

Issues Related to the Quantification of Performance Evaluation Results

This chapter covers issues that are related to the quantification of performance evaluation results. In Section 6.1, we discuss issues related to standard datasets, give some guidelines on what qualities a standard dataset should have and focus on the difficulty in achieving representativeness. We also present some existing datasets for mathematical notation recognition, try to explain the lack of standard datasets, and look at some solutions to remedy this. In Section 6.2, we illustrate the difficulty of defining appropriate matching techniques and propose some possible solutions to the problem. In Section 6.3, we cover the difficulty of selecting a metric based on objective criteria, the difficulty of combining different metrics and the necessity to adapt the metrics. Lastly, in Section 6.4, we cover the importance of algorithm parameters in the evaluation of performance.

6.1 Groundtruthed Dataset Availability

Performance evaluation of mathematical recognition systems often involves providing test data and comparing the algorithm output with the output expected for the test data.

Issue 8: *Standardized datasets are needed for effective performance evaluation.*

Using standardized datasets has many advantages when evaluating the performance of a system. First, it allows the experiments to be reproduced, to validate or negate the results. Second, evaluation of different systems on the same dataset gives results that are more directly comparable [17, 42]. Third, it avoids the creation of custom datasets which could be tuned to a particular algorithm [22].

In the rest of this section, we will discuss desired qualities of datasets, existing datasets, reasons for the limited number of datasets, and use of synthetic data.

6.1.1 Desired qualities of datasets

There are some basic qualities a standard dataset should have in order to support effective performance evaluation.

1 **Datasets should be groundtruthed [42]**, which means that each input is annotated with its expected output. For instance, the image of the mathematical expression x^2 could be annotated with the LaTeX ground truth "`x^2`".

   (a) More than one ground truth can annotate a given input. This is because mathematical notation spans over different levels (see Issue 4.1) and has

different computer representations that can be used (see Issue 5). For example, an image of the expression $g(x+1)$ could be annotated with ground truth at the structure level (represented as the text string "`g(x+1)`"), as well as at the symbolic algebra level (represented as the text string "`g(x+1)`" if g was a function, or as the string "`g*(x+1)`" if g was a multiplication).

(b) Ground truth files should be error free. Errors in the ground truth add noise to the performance measure. Some authors use multiple data entry to reduce errors in the ground truth. Several people independently define ground truth and combine their definitions to create the final ground truth. This technique is described by Valveny *et al.* and was used for creating the University of Washington database [34, 42]

2 Datasets should be publicly available [42]. For instance, the UW-III [30] dataset is sold on a CD-Rom and the Infty dataset can be freely downloaded on the authors website [2].

3 Datasets should be representative of a specified domain. As pointed out by Guida and Mauri [18], it is impossible to compute the real performance of a system since it would require testing it on every expressions of the domain. As a result, the performance measure is only an approximation of the actual one. In order to have a good approximation, the dataset must be representative.

Delineating a domain of mathematical notation is difficult, which leads to this issue:

Issue 8.1: *Datasets should be representative of a known domain, but domains are difficult to characterize and the representativeness of a dataset is difficult to assess.*

The main difficulty is, as pointed out in Section 5.1, that the domain of mathematical notation is not well defined. There are datasets focused on different branches of mathematics, e.g. datasets of arithmetic expressions or datasets of calculus expressions. Unfortunately, we cannot characterize how well they represent the covered domain and its description is generally informal. Ideally, a dataset would have documentation describing the degree of coverage. However, in practice, datasets do not have such characterization and users look at the content to form their own informal impression of the coverage.

There is a need for more precise ways of characterizing a domain. Definition of representativeness must be based on the knowledge of some characteristics of the notation and of the publication corpus. Examples of such characteristics include the frequency of a given operator or the prevalence of a certain document distortion. This knowledge is unfortunately totally or partially unavailable. As a result, it is impossible to define how representative a dataset is.

The best we can do to obtain a representative dataset is to ensure diversity when collecting the data. Different authors give different guidelines on characteristics that should be sought after. Valveny *et al.* say that we should look for different acquisition parameters, deformation, noise and degradation [42]. Garain and Chaudhury talk about the relative frequency of equations per page, the nature of expressions (i.e. varying geometric structures and layouts), variations in typeset methods, page layout, aging effect and other degradations [17]. According to Mao and Kanungo and Valveny

et al. the best mean to attain representativity is to have large datasets [26, 42].

6.1.2 Existing Datasets

There is a lack of standardized datasets and many authors define their own. Tables 6.1 and 6.2 present existing datasets for mathematical notation recognition. They show that the six papers surveyed all use different datasets. However, it takes time for a dataset to be considered as a standard and it is possible that in the following years, some datasets will impose themselves as a standard.

Among the datasets listed in Tables 6.1 and 6.2, some could become standards even though they do not possess all the desired qualities. We have looked at the three datasets that are available. Infty seems to be a good candidate to be a standard dataset. It is quite big with 476 groundtruthed pages and it contains ground truth in MathML, LaTeX and IML. However, when we looked at the data, we observed that the document images do not feature full document pages, but rather disconnected words and equations, which is obviously not representative of what would be found in real situations. On the other hand, the University of Washington third database (UW-III) features full pages that include mathematical expressions. This is a large dataset of document images but only 25 pages contain mathematical expressions and the ground truth does not feature many formats. Garain and Chaudhuri's dataset would also be a good candidate since it is quite large (400 pages and 5,560 expressions). However, implementation of this proposed dataset is in the early stages. The version at URL `http://www.isical.ac.in/~utpal/index.html` contained only 103 expressions which are isolated and not part of a full document page and five parts of document pages including expressions. Also, we observed that the ground truth is

Reference	**Origin**	**Availability**	**Dataset Composition**	**Ground truth Acquisition**	**Ground truth composition**
Ashida *et al.* [7]	*Archiv der Mathematik* *Commentarii Mathematici Helvetici*	Not available due to copyright on the digital libraries	**Symbol Recognition:** 1,400 pages (43,495 typeset expressions) **Structure Analysis:** 700 pages (21,472 typeset expressions)	Automatic recognition and manual correction	Expression/symbol bounding boxes and labels. Expression structure (in extended MathML format)
Chan and Yeung [13]	CRC Standard Mathematical Tables and Formulae [49]	No indications on availability	600 handwritten expressions (11,190 symbols)	Expressions are written by 10 different writters.	Not specified
Garain and Chaudhuri [17]	Various printed and electronic publications	Available at `http://www.isical.ac.in/~utpal/index.html`	400 pages (297 real data, 103 synthetic data) 5,560 typeset expressions	Automatic recognition and manual correction	**For isolated expressions:** LaTeX and symbol bounding boxes **For document pages:** extended MathML format

Table 6.1: List of datasets for mathematical notation recognition.

Reference	Origin	Availability	Dataset Composition	Ground truth Acquisition	Ground truth composition
Takiguchi *et al.* [39]	*Fundamental Formulas of Physics* [27]	Not specified	15 typeset expressions (289 symbols)	Not specified	Not specified
Phillips [34] (U. of Washington Databases)	Various Sources	Available on CD-Rom	Mathematical Content: 25 pages (approx. 100 typeset expressions)	Double entry and triple verification.	For the math expressions: LaTeX and expression/symbol bounding boxes and labels (in Xfig format).
Suzuki *et al.* [37] (Infty)	30 English articles on pure mathematics	Available at `http://www.inftyproject.org`	467 pages (21,056 typeset expressions)	Manual ground truthing	LaTeX , MathML and IML

Table 6.2: List of datasets for mathematical notation recognition (Continued).

not error free: the first two expression images had erroneous ground truth. However, not all of them contain error and we have observed some that had correct ground truth. Nevertheless, in spite of the many problems these three datasets have, they are a good starting point.

6.1.3 Reasons for the Limited Number of Datasets

There are different reasons that can explain the limited number of standard datasets. First, copyright considerations sometimes prevent the authors of dataset from making them available. For instance, in [7], Ashida *et al.* create a dataset (See table 6.1) that they cannot release because some articles in it are protected by copyrights. Second, the creation of a groundtruthed and error free dataset is hard. The groundtruthing process is labour intensive [42, 43]. It is error prone [42, 43] and errors can easily be missed when doing visual inspection. In some cases, the ground truth definition can be subjective [43]. Third, data collection and groundtruthing are labor intensive which makes the acquisition process costly. Baird and Casey report for instance that the three CD-Rom of the University of Washington database cost about $2M [8, 30].

6.1.4 Use of Synthetic Data

As we have seen above, there are many problems that arise during the acquisition of data. One common way to avoid such a problem is to use synthetic data. This means that samples are created using a generation language, LaTeX for example. Since the data is created from a generation language, the ground truth already exists. As a result, it is much more affordable to create a dataset using this type of data. Unfortunately, it is difficult to generate representative datasets. It is also hard to generate

real looking data, and synthetic data will tend not to include as much variations as the real data.

Baird and Casey introduce another solution to make the creation of datasets easier [8]. They propose to create synthetic data by modifying real data through interpolation of the parameters in order to have a larger and more representative dataset. Parameters that they suggest to interpolate are font size, gutter widths, line spacing, resolution, degradation, stroke width and serif length. This method is useful, but has the limitation of not extending the domain that is covered; it requires the acquisition of a good real dataset as a start point.

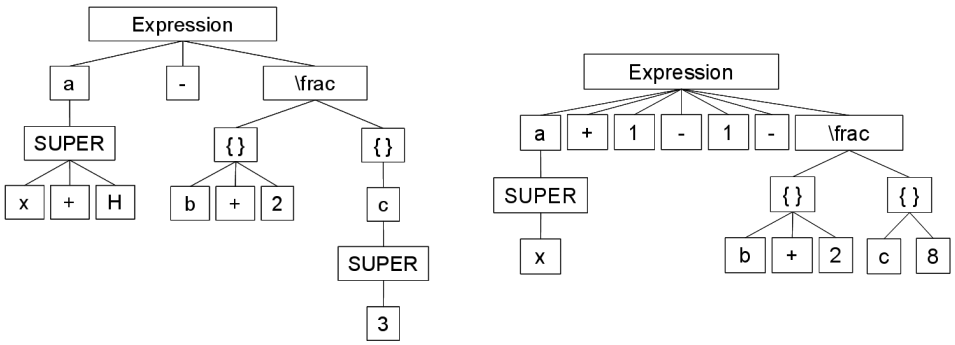
6.2 Groundtruth Matching

Performance of mathematical recognition systems can be evaluated in many different ways, which often involve comparing the systems output with the ground truth.

Issue 9: *Performance evaluation often involves matching output to ground truth. Appropriate matching techniques are difficult to define.*

The matching process is not always straightforward. Matching gets complicated when there are many characters that are part of a complex structure and when there are recognition errors. Figure 6.1(a) shows a tree representation of the ground truth of $a^{x+H} - \frac{b+2}{c^3}$. Figure 6.1(b) shows a hypothetical output from a mathematical recognition system for the same expression. This output contains errors both in the character recognition results and the structure analysis. In this case, the matching can hypothesise that the $+1 - 1$ following the a in the output corresponds to the superscripted $+H$. We can also recognize that the 8 following the c at the denominator

of the fraction is in fact the superscripted 3 of the ground truth. Unfortunately, it is difficult to create automatic matching algorithms that are capable of making such hypotheses.



(a) (b)

Figure 6.1: Illustration of the matching problem. (a) Tree representation of the ground truth of $a^{x+H} - \frac{b+2}{c^3}$. (b) Tree representation of an hypothetical output.

The literature on mathematical notation recognition contains little discussion of the matching issue. Some authors such as Ashida *et al.* make a mention of the matching process, but do not give details on how it is performed [7]. For instance, Ashida *et al.* say that the superfluous symbols are ignored in the comparison process, but they do not say what happens if symbols are missing or the structure misrecognized.

On the other hand, it is possible to find descriptions of the matching issue in other fields of document image analysis. In performance evaluation of document segmentation, Das *et al.* propose to solve the issue of matching the segmented regions with their corresponding ground truth by using graph matching methods [15]. Feng and Manmatha describe a solution that uses an HMM-based alignment algorithm

for matching OCR results and ground truth [16]. Their technique can match words and character over the whole length of a book. Phillips and Chhabra describe the matching of graphics recognition results with the ground truth using match score tables [33]. Another possible solution is to use methods that have been developed for tree and graph isomorphisms, as described by Valiente [41].

6.3 Quantification

The next step after matching the output with the ground truth is to quantify system performance. To do so, metrics must be chosen. Many metrics are described in the literature. Tables 6.3 and 6.4 on pages 56 and 57 show some metrics that can be used for the evaluation of mathematical recognition systems.

Issue 10: *It is difficult to evaluate the quality of performance metrics and to select the best metrics for quantifying system performance.*

6.3.1 Selection of Performance Metrics

The choice of the metric is influenced by the levels of mathematical notation, the aspects of the system one wants to evaluate and the type of usage one wants to make of the mathematical recognition system.

The levels which a system spans can dictate the choice of metric one would use to report performance. For example, we would report the performance of a system that only performs character recognition with symbol recognition rate, recall or precision. On the other hand, we would use the operator recognition rate, baseline recognition rate and symbol placement rate for a system that performs structure analysis.

Different aspects of mathematical recognition systems require different metrics to report them. If we were looking at correctness, for instance, we would use precision, recognition rate and recall to report the performance of the system. If we were looking at efficiency, we would report the computation time.

The type of usage one will make of the system also influences the type of metrics to use. We mean by type of usage whether one is a user or a developer. A user would want to evaluate in order to choose the best system and a developer would use evaluation to give indications in order to modify, improve or create a new system. Users and developers would respectively be more prone to use black box and white box evaluation (see section 3.1) and as a result the corresponding metrics. So, for instance, a user could use recall and precision and a developer historical recall and historical precision.

However, it is not always clear which metric should be used in a given situation. We need ways to evaluate metrics. Some metrics are similar or equivalent, and quantify the same portions or aspects of a system. For instance, operator recognition rate, as reported by Chan and Yeung, measures the structure analysis part of a system [13]. Structure analysis can also be quantified by the baseline recognition rate and symbols placement rate from Zanibbi *et al.* [45]. Evaluation criteria for metrics can be based on the desired qualities of metrics.

1 **Metrics should be objective driven [43]**. This means that the metrics really measure the extent of fulfilment of the system's objectives. For instance, an objective-driven metric for character recognition is the character recognition rate.

2 **Metrics should be comprehensive** [**43**]. It is very seldom that a mathematical recognition system will only have one objective. For instance, a mathematical character recognition system will aim at a high recognition rate and a low false-alarm rate. A comprehensive metric is one that encompasses all the objectives of a system.

3 **Metrics should be compatible with human evaluation** [**43**]. If a system's output is evaluated by a human as excellent, this should be reflected in the metric. This consideration can be used to help guide the development of metrics.

4 **Metrics should be known and accepted by the community** [**42, 43**]. A metric that is not known is not likely to become a standard. The use of standard metrics is highly desirable, as it makes comparison of results easier. Acceptance by the community is also necessary for a metric to be a standard. For instance, recognition rate is known and accepted by the research community as an indicator of the performance of character recognition.

5 **Metrics should be clearly defined to allow reproduction of the evaluation**. As pointed out in issue 2.2, each component of the performance evaluation must be well defined, in order to allow replication of the experiment.

This is not a comprehensive list of qualities of metrics but it is a useful start for evaluation and selection of metrics.

6.3.2 Use of Several Metrics

It is difficult to find a single metric that fulfills all the desired qualities and that can quantify all aspects of a mathematical recognition system. Thus, it is common to use

more than one metric.

> **Issue 10.1:** *Different metrics reflect various aspects of performance. It is not clear how best to combine metrics.*

There are many aspects that might have to be taken into account: correctness, efficiency, usability, comprehensiveness of coverage, quality of coverage, robustness to noise, etc. As pointed out by Valveny *et al.* and by Calingaert, many metrics are often required to capture all these aspects [11, 42].

Multiple metrics can be used in parallel, or they can be combined into a global metric. When metrics are used in parallel, their values are reported as different facets of a system. When metrics are combined, the value of the global metric characterizes the different facets of a system. As we show in Tables 6.3 and 6.4, many authors try to define a metric that characterizes the different aspects of a system's performance.

As pointed out by Wenyin and Dori, combination of metrics is a complex issue [43]. A common practice for combining metrics is to use weighted sums. Unfortunately, the weights are subjective and can vary from one author to the other. This is undesirable since we want performance evaluation to be as objective as possible.

6.3.3 Adaptation of the metrics

As seen throughout this thesis, there is a variety of different mathematical recognition systems that output in different format and at different levels.

> **Issue 11:** *Performance metrics cannot always be applied directly to the output produced by a recognition system. Sometimes the metrics need to be modified to adapt to the type of output.*

Junker *et al.* give an example of adaptation of the definition of metrics [20]. They define recall and precision in four different cases: character/word recognition, structure recognition, document categorization and information extraction.

Similarly, we tested some metrics on a mathematical recognition system that performs structure analysis (DRACULAE) [4]. In order to evaluate the performance of the system, we adapted the definition of precision and recall as given by Valveny *et al.* and by Junker *et al.* (see tables 6.3 and 6.4). We decided to use the concept of baseline as defined by Zanibbi *et al.* [45]. Our adapted definition of precision is the number of correctly recognized baselines over the number of recognized baselines. The adapted definition of recall is the number of correctly recognized baselines over the actual number of baselines. This turns out to be the same as the baseline recognition rate as defined by Zanibbi *et al.*

6.4 Optimization of System Parameters

Optimization of system parameters is an important factor in quantifying performance. Most recognition systems have parameters that can be tuned. If the parameters are left at their default values, performance may not be as good as in the case where the parameters have been tuned.

Issue 12: *Performance evaluation is affected by the setting of system parameters. It is difficult to determine if parameter settings are optimal or not.*

Whether one system is reported to have better performance than another system depends on how the parameters of the two systems are set. A system that has few parameters is preferable, because such a system requires less tuning, and such a

system is likely to have reasonably good performance over a wide range of inputs. A system with many parameters can perhaps be tuned to perform well on a particular type of input, but tuning is time-consuming and performance is likely to fall off greatly when other types of input are presented. Thus, in cases where evaluation is based on the best performance attainable using optimal parameter settings, the number of parameters must be clearly reported. A system that has lower performance, but a small number of parameters, may be preferable to a system that has higher performance but only if a large number of parameters are set properly.

Metric	Reference	Definition
Symbol Recognition Rate	Ashida *et al.* [7]	Number of correctly recognized symbols over the number of symbols.
	Chan and Yeung [13]	
	Suzuki *et al.* [36]	
	Takiguchi *et al.* [38]	
Operator recognition rate	Chan and Yeung [13]	Number of correctly recognized operators over the total number of operators. Used by the authors to measure structural analysis capacities.
Baseline Recognition Rate	Zanibbi *et al.* [45]	Number of correctly recognized baselines over the total number of baselines.
Symbol Placement Rate	Zanibbi *et al.* [45]	Number of symbols that are placed on the correct baseline over the total number of symbols.
Expression Recognition Rate	Zanibbi *et al.* [45]	Number of correctly recognized expressions over the total number of expressions.
	Chan and Yeung [13]	
	Suzuki *et al.* [36]	
Historical Recall	Zanibbi *et al.* [46]	Number of correct hypotheses and false rejections over the total number of entities in the ground truth.
Historical Precision	Zanibbi *et al.* [46]	Number of correct hypothesis and false rejections over the total number of accepted hypothesis and rejected hypothesis.
Recall	Valveny *et al.* [42]	Number of correctly recognized and localized symbols over the number of symbols.
	Junker *et al.* [20]	Number of correctly recognized entities (character/word, structure , document) over the total number of entities.

Table 6.3: List of metrics.

Metric	Reference	Definition
Precision	Valveny *et al.* [42]	Number of correctly localized symbol regions over the total number of localized symbol regions.
	Junker *et al.* [20]	Number of correctly recognized entities (character/word, structure , document) over the number of recognized entities.
Computation time	Valveny *et al.* [42]	Time required to recognize and localize a symbol
	Kosmala *et al.* [23]	Time required to recognize a mathematical expression
Global score	Valveny *et al.* [42]	A combination of recall(r) and precision(p) using the following equation: s=2/[(1/p)+(1/r)]
Global performance index	Garain and Chaudhuri [17]	A global metric that takes into account symbol recognition and arrangement.
Average performance index	Garain and Chaudhuri [17]	Average of the global performance index on all the expressions.
Integrated Performance Measure	Chan and Yeung [13]	Number of correctly recognized symbols and operators over the total number of symbols and operators.
Tree Edit Distance	Valiente [41]	Minimal cost to transform (insert, delete, substitute) a tree into another one.
Post-editing cost	Phillips [33]	The cost estimate for human post-editing effort to clean-up the recognition result. Weighted sum of the number of false alarms, misses, merges and splits.

Table 6.4: List of metrics (Continued).

Chapter 7

Conclusion

In this thesis, we reported a list of issues that arise when trying to evaluate the performance of mathematical recognition systems. In Section 7.1, we discuss the applicability of the listed issues to other areas of document image analysis and look at our contributions to the field of performance evaluation. We conclude this thesis in Section 7.2, with a discussion of future work that could help to address these issues.

7.1 Discussion

Our major contribution to the field of performance evaluation in this thesis is a list of issues that arise when trying to evaluate the performance of mathematical recognition systems. Most of these issues have been mentioned in the literature, but in a scattered fashion. By providing a comprehensive presentation of these issues, we lay the groundwork for a fuller understanding of the complexities of performance evaluation, and for systematic planning of research in the field. For reference, we repeat the list of issues we discussed in this thesis. **Issues related to the reported**

performance measure:

Issue 1: System performance is not always fully reported in the literature.

Issue 1.1: Reported performance measures are hard to compare between systems.

Issue related to the reported evaluation:

Issue 2: Performance evaluation techniques that are used in the literature are not always reported in enough detail to allow reproduction of the experiment.

Issue related to code availability:

Issue 3: Third-party performance evaluation depends on the availability of the source code and the executable code.

Issues related to defining and taking into account the levels of mathematical notation:

Issue 4: The levels of mathematical notation are central to the evaluation process, but they are poorly defined.

Issue 4.1: Performance evaluation must take into account that mathematical expressions are represented at various levels.

Issue related to the various equivalent representations within a format:

Issue 5: Performance evaluation must take into account the various equivalent representations within the computer mathematical formats.

Issues related to the coverage:

Issue 6: Performance evaluation methods should reflect the systems coverage.

Issue 7: It is difficult to determine the extent of coverage of a mathematical recognition system.

Issues related to the lack of standard dataset and their representativeness:

Issue 8: Standardized datasets are needed for effective performance evaluation.

Issue 8.1: Datasets should be representative of a known domain, but domains are difficult to characterize and the representativeness of a dataset is difficult to assess.

Issue related to matching systems output with the corresponding ground truth:

Issue 9: Performance evaluation often involves matching output to ground truth. Appropriate matching techniques are difficult to define.

Issues related to the definition, the choice of metrics, and their combination:

Issue 10: It is difficult to evaluate the quality of performance metrics and to select the best metrics for quantifying system performance.

Issue 10.1: Different metrics reflect various aspects of performance. It is not clear how best to combine metrics.

Issue 11: Performance metrics cannot always be applied directly to the output produced by a recognition system. Sometimes the metrics need to be modified to adapt to the type of output.

Issue related to the parameter settings:

Issue 12: Performance evaluation is affected by the setting of system parameters. It is difficult to determine if parameter settings are optimal or not.

Although our analysis of these issues focused on performance evaluation in the field of mathematical notation recognition, these issues also arise in other areas of document image analysis.

Issues related to the reported performance measure (Section 2.1) are mentioned by Phillips and Chhabra in graphics recognition. They say that the performance of graphics recognition systems are either unknown or only partly reported [33]. Our discussion of issues 1 and 1.1 contributes to their understanding by pointing at the problem of the lack of reported performance measures and by showing that even when these measures are available, comparison of reported performance measures is not easy.

The issue related to the reported evaluation techniques (Section 2.2) also arises in other areas of document image analysis. For instance, Zanibbi *et al.* point out that for table recognition algorithms, the table models are often not explicitly stated in the literature; this affects the reproducibility of the results [47]. Our discussion of issue 2 demonstrates the effect of the lack of details when reporting evaluation techniques on the reproducibility of performance evaluation experiment. Moreover, we proposed guidelines based on Haralick's guidelines to avoid problems when reporting evaluation techniques [19].

The issue of code availability (Chapter 3) in table recognition is mentioned by Zanibbi *et al.*. They point out that the algorithms are often only informally stated in the literature [47]. Similarly, Keogh *et al.*, in data mining, say that code is rarely described well in the literature [22]. Our discussion of issue 3 contributes to the understanding of this issue by showing the different cases of code availability and by discussing the adapted evaluation techniques.

The issues in defining and taking into account the levels of mathematical notation (Section 4.1) are found in other areas of document image analysis. Blostein *et al.* investigate different problems related to the levels of representation in document image analysis. They discuss the difficulty of defining such levels and point at different references in document image analysis where the levels are used [10]. Our contributions to the field of performance evaluation when studying issues 4 and 4.1 are to identify three possible strategies in defining the levels and to present aspects that must be considered when evaluating systems spanning multiple levels.

The issue of the various equivalent representations within a format (Section 4.2) also arises in other areas of document image analysis. We showed in Section 4.2, that there are many equivalent ways to encode mathematical expressions in LaTeX. This is also true for document images since they can be encoded in LaTeX. For instance, the string "*Hello World!*" can be encoded as "`\emph{Hello World!}`" or "`\texit{Hello World!}`" with the same visual output. Our discussion of issue 5 shows the effect of the existence of many representations on performance evaluation. We also contribute to the field of performance evaluation by investigating the use of canonical form to solve the problem.

Issues related to the coverage (Chapter 5) also arise in other fields of document image analysis. OCR systems, for instance, are designed to handle a specific set of characters, a specific language, and they are either for handwritten or for typeset characters. Consequently, not all OCR systems cover the same domain. Our discussion of issues 6 and 7 sheds light on the importance of taking into account the differences in coverage when comparing systems. We also discuss different techniques to determine systems coverage and the difficulties related to each of them. Such a discussion was not found in the literature.

Issues related to the lack of standard datasets and their representativeness (Section 6.1) are mentioned by many authors in the field of document image analysis [8, 19]. Das *et al.* mention it in document image segmentation [15]. In OCR, Feng and Manmatha say that groundtruthed books are rare [16]. Wenyin and Dori talk about the lack of groundtruthed data in graphics recognition [43]. The problem of representativeness of these dataset is mentioned by Baird and Caissey [8]. Our discussion of issues 8 and 8.1 provides an overview of the importance of standard datasets in performance evaluation.

We find instances of the issue of matching system's output with the corresponding ground truth (Section 6.2) in various areas of document image analysis. In graphics recognition, Wenyin and Dori point at the central role of matching algorithms and state that it is not clear how to match and what can be considered as a match [43]. In document segmentation, Das *et al.* talk about the difficulties of comparing the results of different algorithms. They present different existing techniques and propose a graph-isomorphism based technique to compare the results [15]. In OCR, Feng and Manmatha discuss the problem of matching recognizer output with the ground truth

of large documents such as books [16]. In page segmentation, Peng *et al.* propose a region matching technique for the automatic evaluation of different algorithms [32]. Our discussion on issue 9 show that matching is an issue in performance evaluation. We also show that not many papers discuss this issue and that more research is needed.

Issues related to the definition, the choice of metrics, and their combination (Section 6.3) arise in other areas of document image analysis and are mentioned by many authors [18, 19, 20, 42]. In graphics recognition, Phillips and Chhabra propose to measure performance by the amount of work one would need to correct the results [33]. In symbol recognition, Valveny *et al.* propose different qualities of metrics. In graphics recognition, Wenyin and Dori point out that quantification of results is hard and that the combination of metrics is problematic [43]. Junker *et al.* talk about the adaptation of some metrics depending on the document image analysis system that is evaluated [20]. Our discussion on issues 10, 10.1, and 11 provides an overview of the effect of the difficulties of evaluating, combining and adapting metrics on performance evaluation.

The issue of the parameter settings (Section 6.4) arises in other areas of document image analysis. It is mentioned by Mao and Kanungo as a problem for the evaluation of page segmentation. They include the optimization of parameters as a step in their performance evaluation methodology [26]. Outside of document image analysis, Keogh *et al.* talk about the difficulties of comparing and reproducing the results when many parameters are used in data mining algorithms. They also suggest a parameter-free data mining algorithm which might help to solve the issue in document image analysis [22]. Our discussion on issue 12 shows the advantage of using few parameters

and the influence of optimization on performance evaluation.

In summary, we define and discuss the different issues in Chapters 2 to 6. We discuss the impact of each issue on performance by giving an overview of the state of the art for addressing each one of them; and by pointing out open problems. Even though performance evaluation of document image analysis has been studied in many papers, no single place lists all these issues. More work needs to be done, but we believe that our list of issues and their respective discussion will be a useful start for researchers who want to tackle the problem of performance evaluation in the field of document image analysis.

7.2 Future Work

Throughout this thesis, we have listed many issues and proposed some solutions. However, except in the case of the determination of coverage and the definition of a canonical form for LaTeX mathematical expressions, we have not implemented any of these solutions. Future work should include further development and implementation of solutions for the listed issues.

The issues related to the reported performances, the reported evaluation techniques and the code availability are hard to address since they depend on the good will of different authors. Nevertheless, evaluation methodologies and reporting protocols as the one proposed by Haralick, Mao and Kanungo are a sensible approach to improve the completeness of authors' reported performance measures [19, 26].

The issues related to the lack of datasets and their representativeness are difficult and time-consuming to address. The creation of a dataset is demanding and representativeness criteria are ill-defined. A possible way to create an error-free dataset

at lower cost would be to use collaborative techniques similar to the ones used on the Internet. The use of synthetic data and artificial expansion of the dataset are other solutions proposed in the literature and that would be worthwhile to investigate [8, 43].

In order to address the matching issue, techniques that are used to match trees could be used since mathematical expressions can easily be represented by trees [13, 23, 45]. It would be worthwhile to investigate solutions that already exist for matching trees and to apply them to the performance evaluation of mathematical recognition systems.

Lastly, the development of a performance evaluation framework that allows testing different algorithms, using different datasets, matching techniques, and metrics would be a useful tool to address the issues. Such a framework would help to assess the quality of datasets, metrics and matching technique, and it would provide a tool for evaluating the performance of different algorithms.

Bibliography

[1] Oxford English Dictionary Online. http://www.oed.com/, Accessed 2008.04.09.

[2] Infty Project. http://www.inftyproject.org/en/index.html, Accessed 2008.04.23.

[3] MathBrush: A Pen-Based Math System. http://www.cs.uwaterloo.ca/scg/mathbrush/, Accessed 2008.04.23.

[4] The Freehand Formula Entry System. http://www.cs.rit.edu/~rlaz/ffes/index.html, Accessed 2008.04.23.

[5] Moody E. Altamimi and Abdou S. Youssef. A More Canonical Form of Content MathML to Facilitate Math Search. In *Extreme Markup Languages 2007*, 2007.

[6] Dominique Archambault and Victor Moço. Canonical MathML to Simplify Conversion of MathML to Braille Mathematical Notations. In *Lecture Notes in Computer Science*, volume 4061, pages 1191–1198. Springer Berlin / Heidelberg, 2006.

[7] K. Ashida, M. Okamoto, H. Imai, and T. Nakatsuka. Performance Evaluation of a Mathematical Formula Recognition System with a Large Scale of Printed

Formula Images. In *Proceedings of Document Image Analysis for Libraries*, April 2006.

[8] Henry S. Baird and Matthew R. Casey. Towards Versatile Document Analysis Systems. In *Lecture Notes in Computer Science*, volume 3872, pages 280–290. Springer Berlin / Heidelberg, 2006.

[9] D. Blostein and A. Grbavec. Recognition of Mathematical Notation. In *Handbook of Character Recognition and Document Image Analysis*, pages 557–582. World Scientific, 1997.

[10] Dorothea Blostein, Richard Zanibbi, George Nagy, and Rob Harrap. Document Representations. In *Proceedings of Fifth IAPR Workshop on Graphics Recognition (GREC 03)*, pages 3–12, July 2003.

[11] Peter Calingaert. System Performance Evaluation: Survey and Appraisal. *Communications of the ACM*, 10(1):12–18, January 1967.

[12] Kam-Fai Chan and Dit-Yan Yeung. Mathematical Expression Recognition: a Survey. *International Journal on Document Analysis and Recognition*, 3(1):3–15, August 2000.

[13] Kam-Fai Chan and Dit-Yan Yeung. Error Detection, Error Correction and Performance Evaluation in On-Line Mathematical Expression Recognition. *Pattern Recognition*, 34(8):1671–1684, August 2001.

[14] Bidyut B. Chaudhuri and Utpal Garain. An Approach for Recognition and Interpretation of Mathematical Expressions in Printed Document. *Pattern Analysis and Applications*, 3(2):120–131, June 2000.

[15] Amit K. Das, Sanjoy K. Saha, and Bhabatosh Chanda. An Empirical Measure of the Performance of a Document Image Segmentation Algorithm. *International Journal on Document Analysis and Recognition*, 4(3):183–190, March 2002.

[16] Shaolei Feng and R. Manmatha. A Hierarchical, HMM-based Automatic Evaluation of OCR Accuracy for a Digital Library of Books. In *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, pages 109–118, June 2006.

[17] Uptal Garain and Bidyut B. Chaudhuri. A Corpus for OCR Research on Mathematical Expressions. *International Journal on Document Analysis and Recognition*, 7(4):241–259, September 2005.

[18] Giovanni Guida and Giancarlo Mauri. Evaluation of Natural Language Processing Systems: Issues and Approaches. In *Proceedings of the IEEE*, volume 74, pages 1026–1035, July 1986.

[19] Robert M. Haralick. Performance Evaluation of Document Image Algorithms. In *Lecture Notes in Computer Science*, volume 1941, pages 315–323. Springer Berlin / Heidelberg, 2000.

[20] Markus Junker, Andreas Dengel, and Rainer Hoch. On the Evaluation of Document Analysis Components by Recall, Precision, and Accuracy. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, page 713, Washington, DC, USA, 1999.

[21] Markus Junker and Rainer Hoch. An Experimental Evaluation of OCR Text Representations for Learning Document Classifiers. *International Journal on Document Analysis and Recognition*, 1(2):116–122, July 1998.

[22] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards Parameter-Free Data Mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 206–215, 2004.

[23] Andreas Kosmala, Stephane Lavirotte, Loïc Pottier, and Gerhard Rigoll. On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, pages 107–110, 1999.

[24] Stephane Lavirotte and Loïc Pottier. Mathematical Formula Recognition Using Graph Grammar. In *Proceedings of the SPIE*, volume 3305, pages 44–52, 1998.

[25] Wendy G. Lehnert and Beth Sundheim. A Performance Evaluation of Text-Analysis Technologies. *AI Magazine*, 12(3):81–94, 1991.

[26] Song Mao and Tapas Kanungo. Empirical Performance Evaluation Methodology and its Application to Page Segmentation Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):242–256, 2001.

[27] D. H. Menzel. *Fundamental Formulas of Physics*. Prentice-Hall, 1955.

[28] Erik G. Miller and Paul A. Viola. Ambiguity and Constraint in Mathematical Expression Recognition. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 784–791, 1998.

[29] George Nagy. Twenty Years of Document Image Analysis in PAMI. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):38–62, January 2000.

[30] University of Washington. UW-III English/Technical Document Image Database. CD-ROM, 1996.

[31] Masayuki Okamoto, Hiroki Imai, and Kazuhiko Takagi. Performance Evaluation of a Robust Method for Mathematical Expression Recognition. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 121–128, 2001.

[32] Liangrui Peng, Ming Chen, Changsong Liu, Xiaoqing Ding, and Jirong Zheng. An Automatic Performance Evaluation Method for Document Page Segmentation. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 134–137, 2001.

[33] Ihsin T. Phillips and Atul K. Chhabra. Empirical performance evaluation of graphics recognition systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):849–870, 1999.

[34] Ihsin T. Phillips, Jaekyu Ha, Robert M. Haralick, and Dov Dori. The Implementation Methodology for a CD-ROM English Document Database. In *Proceedings of the Second International Conference on Document Analysis and Recognition*, pages 484–487, October 1993.

[35] Steve Smithies, Kevin Novins, and James Arvo. A Handwriting-Based Equation Editor. In *Proceedings of the 1999 conference on Graphics interface*, pages 84–91, 1999.

[36] Masakazu Suzuki, Fumikazu Tamari, Ryoji Fukuda, Seiichi Uchida, and Toshihiro Kanahori. INFTY: an Integrated OCR System for Mathematical Documents. In *Proceedings of the 2003 ACM Symposium on Document Engineering*, pages 95–104, 2003.

[37] Masakazu Suzuki, Seiichi Uchida, and Akihiro Nomura. A Ground-Truthed Mathematical Character and Symbol Image Database. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pages 675–679, 2005.

[38] Yusuke Takiguchi, Minoru Okada, and Yasuji Miyake. A Fundamental Study of Output Translation from Layout Recognition and Semantic Understanding System for Mathematical Formulae. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, volume 2, pages 745–749, 2005.

[39] Yusuke Takiguchi, Minoru Okada, and Yasuji Miyake. A Study on Character Recognition Error Correction at Higher Level Recognition Step for Mathematical Formulae Understanding. In *Proceedings of the 18th International Conference on Pattern Recognition*, volume 2, pages 966–969, 2006.

[40] Michael Thulke, Volker Märgner, and Andreas Dengel. A General Approach to Quality Evaluation of Document Segmentation Results. In *Selected Papers from the Third IAPR Workshop on Document Analysis Systems*, pages 43–57, 1999.

[41] Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, Berlin, 2002.

[42] Ernest Valveny, Philippe Dosch, Adam Winstanley, Yu Zhou, Su Yang, Luo Yan, Liu Wenyin, Dave Elliman, Mathieu Delalandre, Eric Trupin, Sébastien Adam, and Jean-Marc Ogier. A General Framework for the Evaluation of Symbol Recognition Methods. *International Journal on Document Analysis and Recognition*, 9(1):59–74, 2007.

[43] Liu Wenyin and Dov Dori. Performance Evaluation of Graphics Recognition Algorithms: Principles and Applications. In *Proceedings of the 14th International Conference on Pattern Recognition*, volume 2, pages 1180–1182, 1998.

[44] Richard Zanibbi. Recognition of Mathematics Notation via Computer Using Baseline Structure. Master's thesis, Queen's University, Kingston (Canada), January 2000. 95 pages.

[45] Richard Zanibbi, Dorothea Blostein, and James R. Cordy. Recognizing Mathematical Expressions Using Tree Transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1455–1467, 2002.

[46] Richard Zanibbi, Dorothea Blostein, and James R. Cordy. Historical Recall and Precision: Summarizing Generated Hypotheses. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, volume 1, pages 202–206, 2005.

[47] Richard Zanibbi, Dorothea Blostein, and James R. Cordy. Explicit Decisions: A Technique for White-box Evaluation of Recognition Algorithms. Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence, 2007.

[48] Lucy Zhang and Richard Fateman. Survey of User Input Models for Mathematical Recognition: Keyboards, Mice, Tablets, Voice. Computer Science Division, University of California.

[49] Daniel Zwillinger. *CRC Standard Mathematical Tables and Formulae*. CRC Press, 30th edition, 1996.

Appendix A

Coverage Experiment Results

In the following, we present the results from the experiments to determine the coverage of FFES/DRACULAE. Each symbol was inputted three times. If it was recognized once, we knew it was covered by the system. Otherwise, we were unable to draw any conclusion.

Symbol Tested	**Coverage**	**Symbol Tested**	**Coverage**
∞	Covered	$\propto$	No conclusion
%	No conclusion	|	No conclusion
&	Covered	@	No conclusion
:	Covered	$	No conclusion
!	Covered	(	Covered
{	Covered	[	Covered
#	Covered	?	Covered
)	Covered	}	Covered
]	Covered	$\sqrt{}$	No conclusion

Table A.1: Coverage of the miscellaneous symbols by FFES/DRACULAE [4] obtained through experimentations.

Symbol Tested	**Coverage**	**Symbol Tested**	**Coverage**
$+$	Covered	$\cup$	Covered
$-$	No conclusion	$\leq$	No conclusion
$\div$	No conclusion	$\geq$	No conclusion
$\times$	No conclusion	$\cdot$	No conclusion
$*$	Covered	$\rightarrow$	Covered
$=$	No conclusion	$\leftarrow$	No conclusion
$\equiv$	No conclusion	$\leftrightarrow$	Covered
$\cong$	No conclusion	$\emptyset$	Covered
$\neq$	Covered	$\neg$	No conclusion
$<$	Covered	$\pm$	No conclusion
$>$	Covered	$\exists$	Covered
$\cong$	No conclusion	$\cap$	Covered
$\forall$	Covered	$\otimes$	No conclusion
∇	Covered	$\oplus$	No conclusion
$\notin$	Covered	$\therefore$	Covered
$\in$	Covered	$\sim$	Covered
$\subseteq$	No conclusion	$\perp$	Covered
$\subset$	No conclusion	$\Rightarrow$	Covered
$\not\subset$	No conclusion	$\Leftarrow$	No conclusion
$\supseteq$	No conclusion	$\Leftrightarrow$	No conclusion
$\supset$	No conclusion	$\Downarrow$	No conclusion

Table A.2: Coverage of the explicit operators by FFES/DRACULAE [4] obtained through experimentations.

Symbol Tested	Coverage	Symbol Tested	Coverage
A	Covered	a	No conclusion
B	No conclusion	b	Covered
C	Covered	c	No conclusion
D	Covered	d	No conclusion
E	No conclusion	e	No conclusion
F	No conclusion	f	No conclusion
G	No conclusion	g	Covered
H	Covered	h	Covered
I	No conclusion	i	Covered
J	Covered	j	Covered
K	Covered	k	No conclusion
L	Covered	l	No conclusion
M	Covered	m	No conclusion
N	Covered	n	Covered
O	No conclusion	o	Covered
P	No conclusion	p	Covered
Q	No conclusion	q	No conclusion
R	No conclusion	r	Covered
S	Covered	s	No conclusion
T	No conclusion	t	No conclusion
U	No conclusion	u	No conclusion
V	No conclusion	v	No conclusion
W	Covered	w	Covered
X	No conclusion	x	No conclusion
Y	No conclusion	y	Covered
Z	Covered	z	No conclusion

Table A.3: Coverage of the Latin alphabet by FFES/DRACULAE [4] obtained through experimentations.

Symbol Tested	**Coverage**	**Symbol Tested**	**Coverage**
α	Covered	Γ	Covered
β	No conclusion	Δ	No conclusion
γ	No conclusion	Θ	No conclusion
δ	No conclusion	Λ	Covered
ε	Covered	Ξ	No conclusion
ζ	Covered	Π	Covered
η	Covered	Σ	Covered
θ	Covered	Φ	No conclusion
ι	No conclusion	Ψ	Covered
κ	No conclusion	Ω	Covered
λ	Covered		
μ	Covered		
ν	Covered		
ξ	No conclusion		
π	No conclusion		
ρ	Covered		
σ	No conclusion		
τ	No conclusion		
υ	No conclusion		
φ	No conclusion		
χ	No conclusion		
ψ	Covered		
ω	Covered		

Table A.4: Coverage of the Greek alphabet by FFES/DRACULAE [4] obtained through experimentations.

Symbol Tested	**Coverage**	**Symbol Tested**	**Coverage**
0	No conclusion	5	No conclusion
1	Covered	6	No conclusion
2	Covered	7	Covered
3	Covered	8	No conclusion
4	Covered	9	No conclusion

Table A.5: Coverage of the Digits by FFES/DRACULAE [4] obtained through experimentations.