# JBIG2 Supported by OCR

Radim Hatlapatka

Masaryk University, Faculty of Informatics,
Botanická 68a, 602 00 Brno, Czech Republic
`208155@mail.muni.cz`

**Abstract.** Digital Mathematical libraries contain a large volume of PDF documents containing scanned text. In this paper we describe how this documents can be compressed and thus provide them more effectively to the users. We introduce a JBIG2 standard for compressing bitonal images such as scanned text and we discuss issues if OCR is used for improving the compression ratio of jbig2enc open-source encoder. For this purpose we have designed API for using OCR in jbig2enc which we describe in this paper together with already achieved results.

**Keywords:** `jbig2enc`, JBIG2, PDF size optimization, compression, DML, OCR, `pdfJbIm`, `DML-CZ`, EuDML

*Smaller is faster and safer too.* (Stephen Adams, Google)

## 1 Motivation

Digital mathematical libraries (DMLs) contain a large volume of documents with scanned text (more than 80% of EuDML is scanned), which are mostly created by scanning older papers which were written and published earlier and their digital versions are already lost. Documents created this way are referred to as retro-born digital documents.

Research in math is influenced greatly by older articles and papers. When something new in math is discovered or researched, it is often based on older papers and discoveries. To make research more comfortable users require easy access to these kinds of documents. Thus DMLs need to provide documents that are easy to both find and access.

One user demand is for quick and easy access to documents. This means not only the ability to find where a document can be downloaded from, but also the ability to access it from the user's computer. The time to access a document is highly dependent on its size, which can be reduced using a good compression method. Documents in DMLs are mostly stored as PDF documents, which is probably the most widely-used document format on the Internet. In PDF, images are stored using various compression methods. One supported method is the JBIG2 compression method, which offers great compression ratios [1].

The bachelor thesis, JBIG2 Compression by Radim Hatlapatka [2], introduced a method for compressing PDF documents using the JBIG2 standard with jbig2enc open-source encoder. This tool has been re-named pdfJbIm [3]. The improvement to jbig2enc introduced in this bachelor thesis improves the compression ratio of jbig2enc on average by a further 10%.

The results of the comparison of this tool with other tools have been published in [6]. In DML 2010, an article was published about the newer version of pdfJbIm and the results achieved with data stored in DML-CZ [7]. We are developing additional improvements of the jbig2enc encoder which use the results of an OCR engine to decide if two symbols are equivalent and thus should be stored only once in the dictionary. If they are found equivalent, the OCR engine can also help to decide which of them should be stored in the dictionary and be used for reconstructing an image when it is decompressed, thereby improving the quality of the image.

In this paper, we introduce the JBIG2 standard (see Section 2) and discuss issues that need to be addressed when OCR is used for compressing images to achieve the best possible results. We focus on issues connected to documents with math (see Section 3) and we describe a jbig2enc interface designed for using an OCR engine (see Section 4). Finally, we show some experimental results achieved when using Tesseract as the OCR engine (see Section 5).

## 2 Introduction to JBIG2

JBIG2 is a standard for compression of bitonal images developed by the Joint Bi-level Image Experts Group. These are images that consist of two colours only (usually black and white). The main area of such documents is a scanned text. JBIG2 was published in 2000 as an international standard ITU T.88 [9] and one year later as ISO/IEC 14492 [1]. It typically generates files that are three to five times smaller than Fax Group 4 and two to four times smaller than JBIG1, which was the previous standard released by the Joint Bi-level Image Experts Group [5]. JBIG2 also supports "perceptually lossless" coding. This is a special kind of lossy compression which causes no visually noticeable loss. Scanned text often contains flyspecks (tiny pieces of dirt) and perceptually lossless coding can help to get rid of the flyspecks and thus increase the quality of the output image.

The content of each page may be segmented into several regions with specific types of data. Most often it is segmented to a text region for text data, a halftone region for halftone images[1] and a generic region for the rest. In some situations, it is better to use the generic region for a specific type of data rather than a specific region such as the halftone region; in other situations, the converse is true.

The JBIG2 encoder segments text regions into components that most often correspond to symbols. For each set of equivalent symbols, one representant is chosen. The representant contains stored bitmap data and each occurrence of the symbol then directs to that representant with information about its position. These symbols must be encoded (both in the dictionary containing representants and also their occurrences). JBIG2 uses modified versions of Arithmetic and

[1] More about halftone can be found at `http://en.wikipedia.org/wiki/Halftone`

Huffman coding. Huffman coding is used mostly by faxes because of its lower computation demands, even though Arithmetic coding produces slightly better results.

JBIG2 supports a multi-page compression used for symbol coding (the coding of text regions). Any symbol that is used on more than one page is stored in a global dictionary. Such symbols need to be stored only once; space needed to store documents is thereby further reduced.

## 3 Specific Aspects of Using OCR in JBIG2 and Jbig2enc

OCR and image compression according to JBIG2 standard are very similar. In both cases, it is necessary to segment an image into components that are further processed. In OCR, there is necessary to detect text blocks and to detect individual symbols in order to be able to recognize them. In JBIG2, it is also necessary to detect text blocks and ideally also to detect individual symbols in order to be able to achieve the maximum compression ratio. There is one main difference between image compression that follows the JBIG2 standard, and OCR. In OCR, there is necessary to provide results even when OCR engine is uncertain, and to have the OCR engine trained to know the symbols contained in the image. When compressing an image with perceptually lossless compression encoder cannot afford errors but it has an advantage in that if it is uncertain about a symbol it can classify it as a new symbol, thereby preventing unwanted errors. It also does not need to know font information in advance.

When rendering text from images using OCR, the most important part is to recognize what is written, not in what format and fonts. This information is also welcome, but it is not the most important. If you want to compress an image there is necessary to differentiate between symbols in different fonts because the font information can be of value to the user. When using OCR in detecting equivalent symbols, it is necessary to take this information into account, and if this information is not provided by OCR itself, it needs to be handled by additional methods.

It is also necessary to take into account that atypical symbols can appear in documents, and they need to be handled correctly as well. From the OCR point of view, math symbols can be considered atypical. It is either possible to use a specialized OCR which handles math such as the Infty Reader [8], or to detect that it is math and process it specially.

## 4 Jbig2enc API for Using OCR

Our API consists of two parts (modules): one represents the data structure holding the results of OCR and one represents methods for running the OCR engine and retrieving its results.

Our goal is to make the API for holding OCR results and the API for using OCR engine as adaptable as possible in order to allow easy interchangeability of OCR engines and thus prevent unnecessary modifications of existing code.

Because jbig2enc is written in C++, our improvement and API also need to be in C/C++. We decided to use an object hierarchy which allows the creation of a common class with required methods; for creating new modules we used an inheritance. A new module using a different OCR engine is easily made by inheriting the relevant class and implementing defined methods. This methods are implemented specifically for the OCR engine that is actually used.

For holding OCR results, we need to allow the storage of additional data specific to the specific OCR engine that can be used to improve the comparison of representants and thus create a specific similarity function which is most suitable for that OCR engine.

Figure 1 shows classes representing the interface for using an OCR engine. On the left are classes representing the module for using an OCR engine and its function. Class `TesseractOcr` is an example of a module which uses Tesseract as the OCR engine. On the right classes holding results of OCR recognition are described. There is a main class for storing just simple structures with representative symbols. For holding OCR results, it is necessary to store additional information such as text recognized by the OCR engine and its confidence level. For this purpose, the class `OcrResult` is crated, which can be extended and thus new classed can easily be created to store additional information provided by the OCR engine.

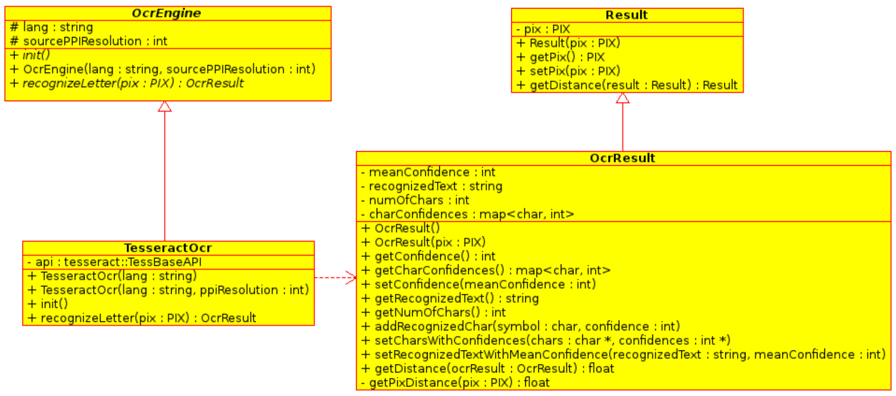

OcrEngine
# lang : string
# sourcePPIResolution : int
+ init()
+ OcrEngine(lang : string, sourcePPIResolution : int)
+ recognizeLetter(pix : PIX) : OcrResult
Result
- pix : PIX
+ Result(pix : PIX)
+ getPix() : PIX
+ setPix(pix : PIX)
+ getDistance(result : Result) : Result
OcrResult
- meanConfidence : int
- recognizedText : string
- numOfChars : int
- charConfidences : map<char, int>
+ OcrResult()
+ OcrResult(pix : PIX)
+ getConfidence() : int
+ getCharConfidences() : map<char, int>
+ setConfidence(meanConfidence : int)
+ getRecognizedText() : string
+ getNumOfChars() : int
+ addRecognizedChar(symbol : char, confidence : int)
+ setCharsWithConfidences(chars : char *, confidences : int *)
+ setRecognizedTextWithMeanConfidence(recognizedText : string, meanConfidence : int)
+ getDistance(ocrResult : OcrResult) : float
- getPixDistance(pix : PIX) : float
TesseractOcr
- api : tesseract::TessBaseAPI
+ TesseractOcr(lang : string)
+ TesseractOcr(lang : string, ppiResolution : int)
+ init()
+ recognizeLetter(pix : PIX) : OcrResult

**Fig. 1.** Jbig2enc API for using an OCR engine

## 5 Experimental Results

In this Section, we introduce the results we achieved using our prototype version of the improved jbig2enc encoder which uses Tesseract as the OCR engine.

In order to show results of the created prototype, we compress set of more than 800 PDF documents. It is a set of PDFs selected randomly from collection of Czech digital mathematical library. Documents are selected in order to cover different types of PDFs from different eras. For this purpose PDFs are chosen from different journals and from papers published in different years.

For compressing PDF documents pdfJbIm [3] is used. It uses our prototype version of the improved jbig2enc encoder. The prototype improves compression ratio by additional two percents in comparison with the previous improvement of the jbig2enc encoder [6].

The achievement is shown in Table 1 and in graph in Figure 2. All shown results are in KB. In order to prevent errors even on documents with an extremely bad quality, default thresholding value[2] is set to minimize potential loss of data. The documents also contain nonbitonal images that are ignored.

**Table 1.** Results of an enhanced jbig2enc encoder

| Number of pages | Original PDF | Original jbig2enc | Improved jbig2enc without OCR | Improved jbig2enc with OCR |
|---|---|---|---|---|
| 1 | 107.11 | 88.64 (82.8%) | 86.72 (81%) | 84.63 (79%) |
| 2 | 240.76 | 203.19 (84.3%) | 198.47 (82.4%) | 193.83 (80.5%) |
| 3 | 353.87 | 296.73 (83.9%) | 288.11 (81.4%) | 281.21 (79.5%) |
| 4 | 476.82 | 401.13 (84.1%) | 388.85 (81.6%) | 379.38 (79.6%) |
| 5 | 592.42 | 499.82 (84.4%) | 484.31 (81.7%) | 472.61 (79.8%) |
| 6 | 722.71 | 609.02 (84.3%) | 590.66 (81.7%) | 576.42 (79.8%) |
| 7 | 822.41 | 691.49 (84.1%) | 667.13 (81.1%) | 650.51 (79.1%) |
| 8 | 949.18 | 800.55 (84.3%) | 775.36 (81.7%) | 756.16 (79.7%) |
| 9 | 1,080.05 | 913.35 (84.6%) | 880.55 (81.5%) | 858.71 (79.5%) |
| 10 | 1,161.09 | 975.56 (84%) | 936.53 (80.6%) | 913.19 (78.6%) |

Figure 3 represents an original image (TIFF G4 compressed) that is further compressed according to the JBIG2 standard. The image size is 118 KB. Figures 4 (size 8.6 KB) and 5 (size 8.2 KB) are images compressed according JBIG2 standard using the jbig2enc open-source encoder [4]. Their sizes are around 8 KB which is a significant reduction from the original image. The difference between Figures 4 and 5 is that Figure 5 is compressed by the improved jbig2enc as described in Section 4. It uses Tesseract as the OCR engine.

In these figures, there is no visible loss of data or image quality, and without searching for differences in detail they look the same. Figure 6 shows the difference between the original image and the image compressed using the jbig2enc encoder which uses Tesseract as the OCR engine. Figure 7 shows how the output image changes when the jbig2enc encoder uses an OCR engine to improve its compression ratio, and as side effect has the potential to improve the quality of the output image.

[2] Thresholding value used by the jbig2enc encoder even if no improvement is used. It determines if two symbols should be considered equivalent or not

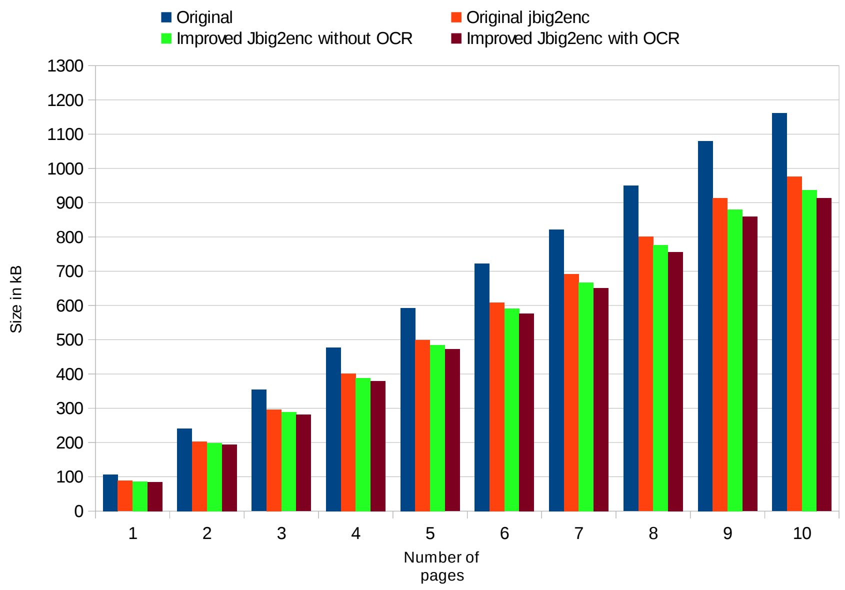


**Fig. 2.** Compression results of jbig2enc and its improved versions

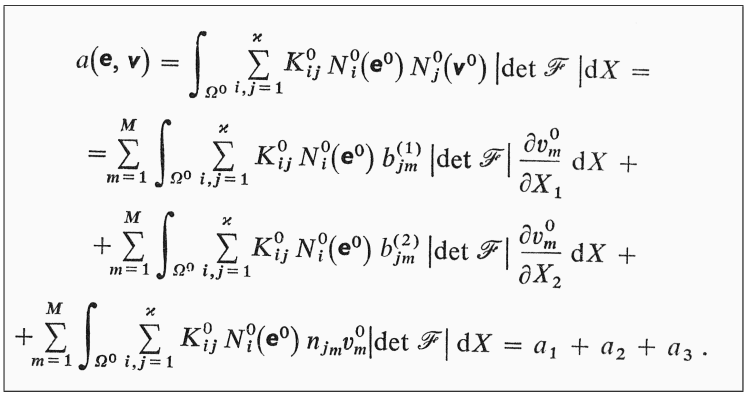
## 6 Conclusion and Future Work

As we have shown, using OCR in jbig2enc further improves its compression ratio. We defined the API, which is independent of the OCR engine used. As a default OCR engine, we used the Tesseract OCR, but this can be easily replaced by another OCR engine by creating a new module specifically for use with that OCR engine which implements the defined API and by changing one line in the existing code to determine which OCR engine is used.

We have shown that by using OCR, we are able to choose which representant of several equivalent ones is better in terms of visual quality, thus improving the quality of the image.

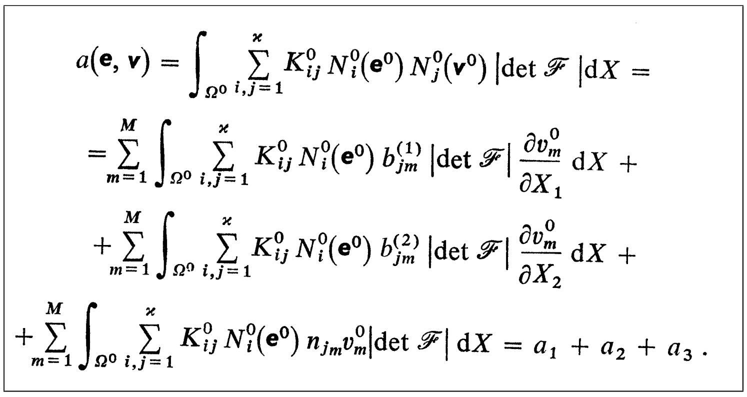
The similarity distance function is not yet fully balanced for simultaneously maximizing the compression ratio and preventing errors. This needs to be solved and well tested. For this purpose, we intend to create semi-automatic testing by experimenting on images for which we know output will be created correctly without quality reduction, and for which we know a recognized number of different symbols. We shall then simulate the decrease in quality caused by scanning the image and see to what extent the result changes.

We intend to create a universal language dictionary containing all of the symbols used in European languages, including math symbols, and train Tesseract for it. This should prevent the need for the user to set all used languages and improve speed if more languages are used. More dictionaries means recurrence of the same symbols, thus creating a bigger collection than necessary. While



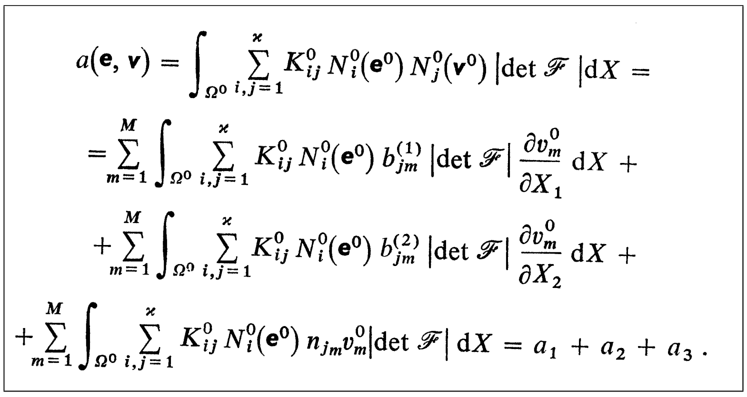
$$
\begin{aligned}
a(\mathbf{e},\mathbf{v}) &= \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, N^0_j(\mathbf{v}^0)\, |\det \mathscr{F}\,|\mathrm{d}X = \\
&= \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, b^{(1)}_{jm}\, |\det \mathscr{F}|\, \frac{\partial v^0_m}{\partial X_1}\,\mathrm{d}X + \\
&+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, b^{(2)}_{jm}\, |\det \mathscr{F}|\, \frac{\partial v^0_m}{\partial X_2}\,\mathrm{d}X + \\
&+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, n_{jm} v^0_m |\det \mathscr{F}|\,\mathrm{d}X = a_1 + a_2 + a_3\,.
\end{aligned}
$$

**Fig. 3.** Original image before JBIG2 compression (TIFF G4 compressed, size: 118 KB)



$$
\begin{aligned}
a(\mathbf{e},\mathbf{v}) &= \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, N^0_j(\mathbf{v}^0)\, |\det \mathscr{F}\,|\mathrm{d}X = \\
&= \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, b^{(1)}_{jm}\, |\det \mathscr{F}|\, \frac{\partial v^0_m}{\partial X_1}\,\mathrm{d}X + \\
&+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, b^{(2)}_{jm}\, |\det \mathscr{F}|\, \frac{\partial v^0_m}{\partial X_2}\,\mathrm{d}X + \\
&+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, n_{jm} v^0_m |\det \mathscr{F}|\,\mathrm{d}X = a_1 + a_2 + a_3\,.
\end{aligned}
$$

**Fig. 4.** Images compressed according to standard JBIG2 without OCR usage (size: 8.6 KB)



$$
\begin{aligned}
a(\mathbf{e},\mathbf{v}) &= \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, N^0_j(\mathbf{v}^0)\, |\det \mathscr{F}\,|\mathrm{d}X = \\
&= \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, b^{(1)}_{jm}\, |\det \mathscr{F}|\, \frac{\partial v^0_m}{\partial X_1}\,\mathrm{d}X + \\
&+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, b^{(2)}_{jm}\, |\det \mathscr{F}|\, \frac{\partial v^0_m}{\partial X_2}\,\mathrm{d}X + \\
&+ \sum_{m=1}^{M} \int_{\Omega^0} \sum_{i,j=1}^{\varkappa} K^0_{ij}\, N^0_i(\mathbf{e}^0)\, n_{jm} v^0_m |\det \mathscr{F}|\,\mathrm{d}X = a_1 + a_2 + a_3\,.
\end{aligned}
$$

**Fig. 5.** Image compressed according to standard JBIG2 with OCR usage (size: 8.2 KB)

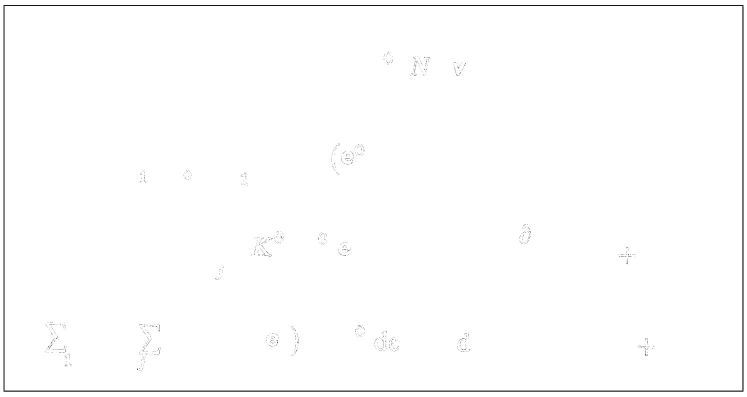


**Fig. 6.** Difference between Fig. 3 and Fig. 5

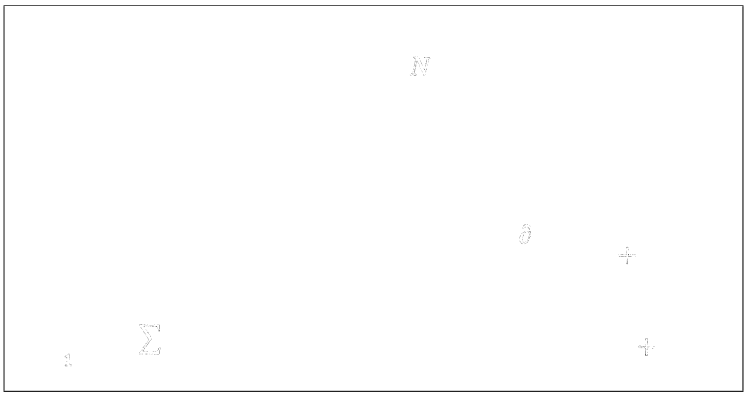


**Fig. 7.** Difference between Fig. 4 and Fig. 5

recognizing symbols, the OCR engine compares data with the existing collection of symbols created based on the dictionaries provided to determine what symbol is represented by the image. The size of this collection influences the amount of comparisons needed to determine which is the best candidate.

There is also a plan for creating a module for using Infty as the OCR engine and to use its math recognition support to improve the compression ratio for documents containing lots of math.

**Acknowledgement**

This work has been financed in part by the European Union through its Competitiveness and Innovation Programme (Information and Communication Technologies Policy Support Programme, "Open access to scientific information", Grant Agreement No. 250503).

## References

1. Committee, J.: 14492 FCD. ISO/IEC JTC 1/SC 29/WG 1 (1999), `http://www.jpeg.org/public/fcd14492.pdf`
2. Hatlapatka, R.: JBIG2 komprese (Bachelor thesis written in Czech, JBIG2 compression). Masaryk University, Faculty of Informatics (advisor Petr Sojka), Brno, Czech Republic (2010)
3. Hatlapatka, R.: PDF Recompression using JBIG2. [online] (2012), `http://nlp.fi.muni.cz/projekty/eudml/pdfRecompression/`
4. Langley, A.: Homepage of jbig2enc encoder. [online], `http://github.com/agl/jbig2enc`
5. SG, S.J.: JBIG Maui Meeting Press Release (December 1999), `http://www.jpeg.org/public/mauijbig.pdf`
6. Sojka, P., Hatlapatka, R.: Document engineering for a digital library: PDF recompression using JBIG2 and other optimizations of PDF documents. In: Proceedings of the 10th ACM symposium on Document engineering. pp. 3–12. DocEng '10, ACM, New York, NY, USA (2010), `http://doi.acm.org/10.1145/1860559.1860563`
7. Sojka, P., Hatlapatka, R.: PDF Enhancements Tools for a Digital Library: pdfJbIm and pdfsign. In: DML 2010 Towards a Digital Mathematics Library. pp. 45–55. Masaryk University, Brno, Czech Republic (2010)
8. Suzuki, M., Tamari, F., Fukuda, R., Uchida, S., Kanahori, T.: INFTY – An Integrated OCR System for Mathematical Documents. In: Proceedings of ACM Symposium on Document Engineering 2003. ACM, Grenoble (2003), `http://www.inftyproject.org/articles/2003_DocEng_Suzuki.zip`
9. Union, I.T.: ITU-T Recommendation T.88. ITU-T Recommendation T.88 (2000), `http://www.itu.int/rec/T-REC-T.88-200002-I/en`