Automatic Extraction of Figures from Scientific Publications in High-Energy Physics

Piotr Adam Praczyk, Javier Nogueras-Iso, and Salvatore Mele

ABSTRACT

Plots and figures play an important role in the process of understanding a scientific publication, providing overviews of large amounts of data or ideas that are difficult to intuitively present using only the text. State-of-the-art digital libraries, which serve as gateways to knowledge encoded in scholarly writings, do not yet take full advantage of the graphical content of documents. Enabling machines to automatically unlock the meaning of scientific illustrations would allow immense improvements in the way scientists work and the way knowledge is processed. In this paper, we present a novel solution for the initial problem of processing graphical content, obtaining figures from scholarly publications stored in PDF. Our method relies on vector properties of documents and, as such, does not introduce additional errors, unlike methods based on raster image processing. Emphasis has been placed on correctly processing documents in high-energy physics. The described approach distinguishes different classes of objects appearing in PDF documents and uses spatial clustering techniques to group objects into larger logical entities. Many heuristics allow the rejection of incorrect figure candidates and the extraction of different types of metadata.

INTRODUCTION

Notwithstanding the technological advances of large-scale digital libraries and novel technologies to package, store, and exchange scientific information, scientists' communication pattern has changed little in the past few decades, if not the past few centuries. The key information of scientific articles is still packaged in a form of text and, for several scientific disciplines, in a form of figures.

New semantic text-mining technologies are unlocking the information in scientific discourse, and there exist some remarkable examples of attempts to extract figures from scientific publications,[1] but current attempts do not provide a sufficient level of generality to deal with figures from high-energy physics (HEP) and cannot be applied in a digital library like INSPIRE, which is our main

Piotr Adam Praczyk (piotr.praczyk@gmail.com) is a PhD student at Universidad de Zaragoza, Spain, and research grant holder at the Scientific Information Service of CERN, Geneva, Switzerland. **Javier Nogueras-Iso** (jnog@unizar.es) is Associate Professor, Computer Science and Systems Engineering Department, Universidad de Zaragoza, Spain. **Salvatore Mele** (Salvatore.Mele@cern.ch) is leader of the Open Access section at the Scientific Information Service of CERN, Geneva, Switzerland.

point of interest. Scholarly publications in HEP tend to contain highly specific types of figures (as any type of graphical content illustrating the text and referenced from it). In particular, they contain a high volume of plots, which are line-art images illustrating a dependency of a certain quality on a parameter.

The graphical content of scholarly publications allows much more efficient access to the most important results presented in a publication.[2,3] The human brain perceives the graphical content much faster than reading an equivalent block of text. Presenting figures with the publication summary when displaying search results would allow more accurate assessment of the article content and in turn lead to a better use of researchers' time. Enabling users to search for figures describing similar quantities or phenomena could become a very powerful tool for finding publications describing similar results. Combined with additional metadata, it could provide knowledge about evolution of certain measurements or ideas over time.

These and many more applications created an incentive to research possible ways to integrate figures in INSPIRE. INSPIRE is a digital library for HEP,[4] the application field of this work. It provides a large-scale digital library service (1 million records, fifty-thousand users), which is starting to explore new mechanisms of using figures in articles of the field to index, retrieve, and present information.[5,6] As a first step, direct access to graphical content before accessing the text of a publication can be provided. Second, a description of graphics ("blue-band plot," "the yellow shape region") could be used in addition to metadata or full-text queries to retrieve a piece of information. Finally, articles could be aggregated into clusters containing the same or similar plots in a possible alternative automated answer to a standing issue in information management.

The indispensable step to realize this vision is an automated, resilient, and high-efficiency extraction of figures from scientific publications. In this paper, we present an approach that we have developed to address this challenge. The focus has been put on developing a general method allowing the extraction of data from documents stored in Portable Document Format (PDF). The results of the algorithm consist of metadata, raster images of a figure, but also vector graphics, which allows easier further processing.

The PDF format has been chosen as the input of the algorithm because it is a de facto standard in scientific communication. In the case of HEP, mathematics, and other exact sciences, the majority of publications are prepared using the Latex document formatting system and later compiled into a PDF file. The electronic versions of publications from outstanding scientific journals are also provided in PDF. The internal structure of PDF files does not always reveal the location of graphics. In some cases, images are included as external entities and easily distinguishable from the rest of a document's content, but other times they are mixed with the rest of the content. Therefore, to miss any figures, the low-level structure of a PDF had to be analyzed. The work described in this paper focuses on the area of HEP. However, with minor variations, the described methods could be applicable to a different area of knowledge.

RELATED WORK

Over years of development of digital libraries and document processing, researchers came up with several methods of automatically extracting and processing graphics appearing in PDF documents. Based on properties of the processed content, these methods can be divided into two groups. The attempts of the first category deal with PDF documents in general, not making any assumptions about the content of encoded graphics or document type. The methods from the second group are more specific to figures from scientific publications. Our approach belongs to the second group.

Tools include command line programs like PDF-Images (http://sourceforge.net/projects/pdf-images/) or web-based applications like PDF to Word (http://www.pdftoword.com/). These solutions are useful for general documents, but all suffer from the same difficulties when processing scientific publications: graphics that are recognized by such tools have to be marked as graphics inside PDF documents. This is the case with raster graphics and some other internally stored objects. In the case of scholarly documents, most graphics are constructed internally using PDF primitives and thus cannot be correctly processed by tools from the first group. Moreover, general tools do not have the necessary knowledge to produce metadata describing the extracted content.

With respect to specific tools for scientific publications it must be noted first that important scientific publishers like Springer or Elsevier have created services to allow access to figures present in scientific publications: the improvement of the SciVerse Science Direct site (http://www.sciencedirect.com) for searching images in the case of Elsevier[7] and the SpringerImages service (http://www.springerimages.com/) in the case of Springer.[8] These services allow searches triggered from a text box, where the user can introduce a description of the required content. It is also possible to browse images by categories such as types of graphics (image, table, line art, video, etc.). The search engines are limited to searches based on figure captions. In this sense, there is little difference between the image search and text search implemented in a typical digital library.

Most of existing works aiming at the retrieval and analysis of figures use the rasterized graphical representation of source documents as its basis. Browuer et al. and Kataria et al. describe a method of detecting plots by means of wavelet analysis.[9,10] They focus on the extraction of data points from identified figures. In particular, they address the challenge of correctly identifying overlapping points of data in plots. This problem would not manifest itself often in the case of vector graphics, which is the scenario proposed in our extraction method. Vector graphics preserve much more information about the documents content than simple values of pixel colours. In particular, vector graphics describe overlapping objects separately. Raster methods are also much more prone to additional errors being introduced during the recognition/extraction phase. The methods described in this paper could be used with Kataria's method for documents resulting from a digitization process.[11]

Liu et al. present a page box-cutting algorithm for the extraction of tables from PDF documents.[12] Their approach is not directly applicable, but their ideas of geometrical clustering of PDF primitives are similar to the ones proposed in our work. However, our experiments with their implementation and HEP publications have shown that the heuristics used in their work cannot be directly applied to HEP, showing the need for an adapted approach, even in the case of tables.

A different category of work, not directly related to graphics extraction but useful when designing algorithms, has been devoted to the analysis of graph use in scientific publications. The results presented by Cleveland describe a more general case than HEP publications.[13] Even if the data presented in the work came from scientific publications before 1984, included observations—for example, typical sizes of graphs—were useful with respect to general properties of figures and were taken into account when adjusting parameters of the presented algorithm.

Finally, there exist attempts to extract layout information from PDF documents. The knowledge of page layout is useful to distinguish independent parts of the content. The approach of layout and content extraction presented by Chao and Fan is the closest to the one we propose in this paper.[14] The difference lies in the fact that we are focusing on the extraction of plots and figures from scientific documents, which usually follow stricter conventions. Therefore we can make more assumptions about their content and extract more precise data. For instance, our method emphasizes the role of detected captions and permits them to modify the way in which graphics are treated. We also extract portions of information that are difficult to be extracted using more general methods, such as captions of figures.

METHOD

PDF files have a complex internal structure allowing them to embed various external objects and to include various types of metadata. However, the central part of every PDF file consists of a visual description of the subsequent pages. The imaging model of PDF uses a language based on a subset of the PostScript language. PostScript is a complete programming language containing instructions (also called operators) allowing the rendering of text and images on a virtual canvas. The canvas can correspond to a computer screen or to another, possibly virtual, device used to visualize the file. The subset of PostScript, which was used to describe content of PDFs, had been stripped from all the flow control operations (like loops and conditional executions), which makes it much simpler to interpret than the original PostScript. Additionally, the state of the renderer is not preserved between subsequent pages, making their interpretation independent.

To avoid many technical details, which are irrelevant in this context, we will consider a PDF document as a sequence of operators (also called the content stream). Every operator can trigger a modification of the graphical state of the PDF interpreter, which might be drawing a graphical primitive, rendering an external attached object, or modifying a position of the graphical pointer[15] or a transformation matrix.[16] The outcome of an atomic operation encoded in the content stream depends not only on parameters of the operation, but also on the way previous operators modified the state of the interpreter. Such a design makes a PDF file easy to render but not necessarily easy to analyze.

Figure 1 provides an overview of the proposed extraction method. At the very first stage, the document is pre-processed and operators are extracted (see "Pre-processing of Operators" below). Later, graphical[17] and textual[18] operators are clustered using different criteria (see "Inclusion of Text Parts" and "Detection and Matching of Captions" below), and the first round of heuristics rejects regions that cannot be considered figures. In the next phase, the clusters of graphical operators are merged with text operators representing fragments of text to be included inside a figure (see "Inclusion of Text Parts" below). The second round of heuristics detects clusters that are unlikely to be figures. Text areas detected by the means of clustering text operations are searched for possible figure captions (see "Detection and Matching of Captions" below). Captions are matched with corresponding figure candidates, and geometrical properties of captions are used to refine the detected graphics. The last step generates data in a format convenient for further processing (see "Generation of the Output" below).

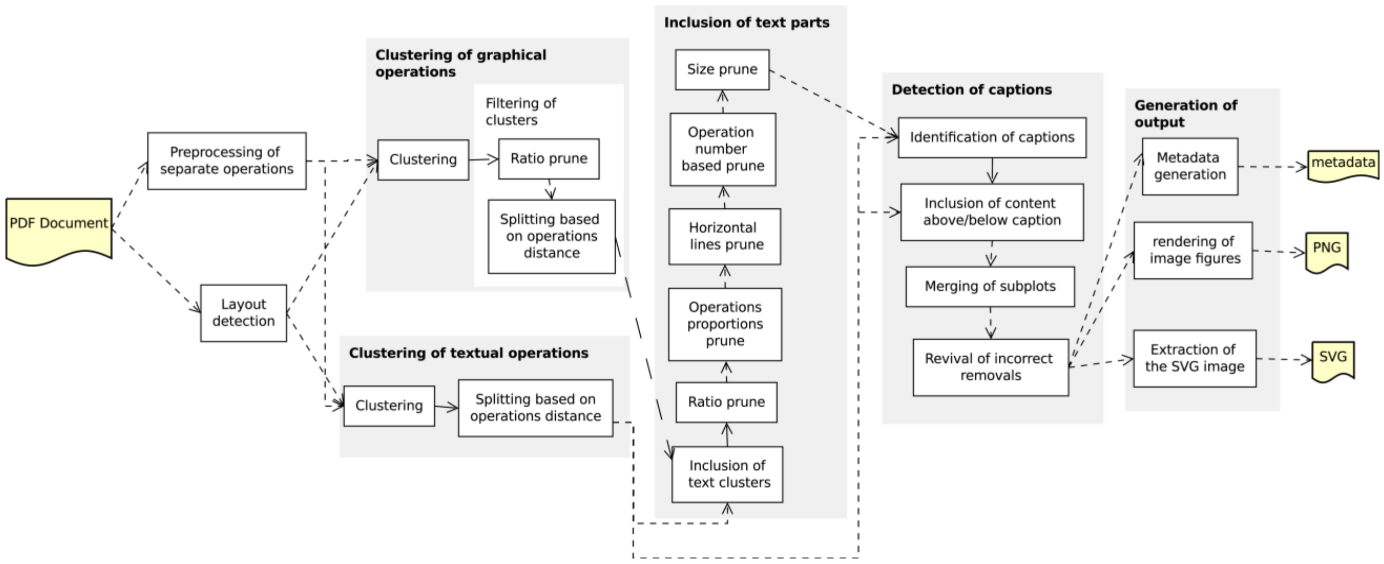

PDF Document
Preprocessing of separate operations
Layout detection
Clustering of graphical operations
Clustering
Filtering of clusters
Ratio prune
Splitting based on operations distance
Clustering of textual operations
Clustering
Splitting based on operations distance
Inclusion of text parts
Size prune
Operation number based prune
Horizontal lines prune
Operations proportions prune
Ratio prune
Inclusion of text clusters
Detection of captions
Identification of captions
Inclusion of content above/below caption
Merging of subplots
Revival of incorrect removals
Generation of output
Metadata generation
rendering of image figures
Extraction of the SVG image
metadata
PNG
SVG

Figure 1. Overview of the figure extraction method.

Additionally, it must be noted that another important pre-processing step of the method consists of the layout detection. An algorithm for segmenting pages into layout elements called page divisions is presented later in the paper. This considerably improves the accuracy of the extraction method because elements from different page divisions can no longer be considered to belong to the same cluster (and subsequently figure). This allows the method to be applied separately to different columns of a document page.

Pre-processing of Operators

The proposed algorithm considers only certain properties of a PDF operator rather than trying to completely understand its effect. Considered properties consist of the operators' type, the region of the page where the operator produces output and, in the case of textual operations, the string representation of the result. For simplicity, we suppress the notion of coordinate system transformation, inherent for the PDF rendering, and describe all operators in a single coordinate system of a virtual 2-dimensional canvas where operations take effect. Transformation operators[19] are assigned an empty operation region as they do not modify the result directly but affect subsequent operations.

In our implementation, an existing PDF rendering library has been used to determine boundaries of operators. Rather than trying to understand all possible types of operators, we check the area of the canvas that has been affected by an operation. If the area is empty, we consider the operation to be a transformation. If there exists a non-empty area that has been changed, we check if the operator belongs to a maintained list of textual operators. This list is created based on the PDF specification. If so, the operators argument list is scanned searching for a string and the operation is considered to be textual. An operation that is neither a transformation nor a textual operation is considered to be graphical. It might happen that text is generated using a graphical operator. However, such a situation is unusual. In the case of operators triggering the rendering of other operators, which is the case when rendering text using type-3 fonts, we consider only the top-level operation.

In most cases, separate operations are not equivalent to logical entities considered by a human reader (such as a paragraph, a figure, or a heading). Graphical operators are usually responsible for displaying lines or curve segments while humans think in terms of illustrations, data lines, etc. Similarly, in the case of text, operators do not have to represent complete or separate words or paragraphs. They usually render parts of words and sometimes parts of more than one word.

The only assumption we make about the relation between operators and logical entities is that a single operator does not trigger rendering of elements from different detected entities (figures, captions). This is usually true because logical entities tend to be separated by a modification of the context—there is a distance between text paragraphs or an empty space between curves.

Clustering of Graphical Operators

The Clustering Algorithm

The representation of a document as a stream of rectangles allows the calculation of more abstract elements of the document. In our model, every logical entity of the document is equivalent to a set of operators. The set of all operators of the document is divided into disjoint subsets in the process called clustering. Operators are decided to belong to the same cluster based on the position of their boundaries. The criteria for the clustering are based on a simple but important observation:

operations forming a logical entity have boundaries lying close to each other. Groups of operations forming different entities are separated by empty spaces.

```
1:  Input: OperationSet input_operations {Set of operators of the same type}
2:  Output: Map<Rectangle, OperationSet> {Spatial clusters of operators}
3:  IntervalTree $t_x$ ← IntervalTree()
4:  IntervalTree ty ← IntervalTree()
5:  Map<Operation, Operation> parent ← Map()
6:  for all Operation op ∈ input_operations do
7:      Rectangle boundary ← extendByMargins(op.boundary)
8:      repeat
9:          OperationSet $int\_ops_x$ ← $t_x$.getIntersectingOps(boundary)
10:         OperationSet $int\_ops_y$ ← $t_y$.getIntersectingOps(boundary)
11:         OperationSet int_ops ← $int\_ops_x \cap int\_ops_y$
12:         for all Operation int_op ∈ int_ops do
13:             Rectangle bd ← $t_x$[int_op] × $t_y$[int_op]
14:             boundary ← smallestEnclosing(bd, boundary)
15:             Parent[int_op] ← op
16:             $t_x$.remove(int_op); $t_y$.remove(int_op)
17:         end for
18:     until int_ops = ∅
19:     $t_x$.add(boundary, op); $t_y$.add(boundary, op)
20: end for
21: Map<Rectangle, OperationSet> results ← Map()
22: for all Operation op ∈ input_operations do
23:     Operation root_ob ← getRoot(parent, op)
24:     Rectangle rec ← $t_x$[int_ob] × $t_y$[int_ob]
25:     if not results.has_key(rec) then
26:         results[rec] ← List()
27:     end if
28:     results[rec].add(op)
29: end for
30: return results
```

Algorithm 1. The clustering algorithm.

The clustering of textual operations yields text paragraphs and smaller objects like section headings. However, in the case of graphical operations, we can obtain consistent parts of images, but usually not complete figures yet. Outcomes of the clustering are utilized during the process of figures detection.

Algorithm 1 shows the pseudo-code of the clustering algorithm. The input of the algorithm consists of a set of pre-processed operators annotated with their affected area. The output is a division of the input set into disjoint clusters. Every cluster is assigned a boundary equal to the smallest rectangle containing boundaries of all included operations.

In the first stage of the algorithm (lines 6–20), we organize all input operations in a data structure of forest of trees. Every tree describes a separate cluster of operations. The second stage (lines 21–29) converts the results (clusters) into a more suitable format.

The clustering of operations is based on the relation of their rectangles being close to each other. Definition 1 formalizes the notion of being close, making it useful for the algorithm.

Definition 1*: Two rectangles are considered to be located close to each other if they are intersecting after expanding their boundaries in every direction by a margin.*

The value by which rectangles should be extended is a parameter of the algorithm and might be different in various situations. To detect if rectangles are close to each other, we needed a data structure allowing the storage a set of rectangles. This data structure was required to allow retrieving all stored rectangles that intersect a given one.

We have constructed the necessary structure using an important observation about the operation result areas. In our model all bounding rectangles have their edges parallel to the edges of the reference canvas on which the output of the operators is rendered. This allowed us to reduce our problem from the case of 2-dimensional rectangles to the case of 1-dimensional intervals. We can assume that edges of the rectangular canvas define the coordinates system. It is easy to prove that two rectangles of edges parallel to the axis of the coordinates system intersect only if both their projections in the directions of axis intersect. The projection of a rectangle into an axis is always an interval.

The observation made above has allowed us to build the required 2-dimensional data structure by remembering two 1-dimensional data structures that recall a number of intervals and for a given interval return the set of intersecting ones. Such a 1-dimensional data structure has been provided by interval-trees.[20] Every interval inside the tree has an arbitrary object assigned to it, which in this case is a representation of the PDF operator. This object can be treated as an identifier of the interval. The data structure also implements a dictionary interface, mapping objects to actual intervals.

At the beginning, the algorithm initializes two empty interval trees representing projections on the X and Y axes, respectively. Those trees store values about projections of the biggest so-far calculated areas rather than about particular operators. Each cluster is represented by the most recently discovered operation belonging to it.

During the algorithm execution, each operator from the input set is considered only once. The order of processing is not important. The processing of a single operator proceeds as follows (the interior of the outermost “for all” loop of the algorithm).

1. The boundary of the operation is extended by the width of margins. The spatial data structure described earlier is utilized to retrieve boundaries of all already detected clusters (lines 9–10)
2. The forest of trees representing clusters is updated. The currently processed operation is added without a parent. Roots of all trees representing intersecting clusters (retrieved in previous step) are attached as children of the new operation.

3. The boundary of the processed operation is extended to become the smallest rectangle containing all boundaries of intersecting clusters and the original boundary. Finally, all intersecting clusters are removed from the spatial data structure.
4. Lines 9–17 of the algorithm are repeated as long as there exist areas intersecting the current boundary. In some special cases, more than one iteration may be necessary.
5. Finally, the calculated boundary is inserted into the spatial data structure as a boundary of a new cluster. The currently processed operation is designed to represent the cluster and so is remembered as a representation of the cluster.

After processing all available operations, the post–processing phase begins. All the trees are transformed into lists. The resulting data structure is a dictionary having boundaries of detected clusters as keys and lists of belonging operations as values. This is achieved in lines 21–29. During the process of retrieving the cluster to which a given operation belongs, we use a technique called path compression, known from the union-find data structure.[21]

Filtering of Clusters

Graphical areas detected by a simple clustering usually do not directly correspond to figures. The main reason for this is that figures may contain not only graphics, but also portions of text. Moreover, not all graphics present in the document must be part of a figure. For instance, common graphical elements not belonging to a figure include logos of institutions and text separators like lines and boxes; various parts of mathematical formulas usually include graphical operations; and in the case of slides from presentations, the graphical layout should not be considered part of a figure.

The above shows that the clustering algorithm described earlier is not sufficient for the purpose of figures detection and it yields a results set wider than expected. In order to take into account the aforementioned characteristics, pre-calculated graphical areas are subject to further refinement. This part of the processing is highly domain-dependent as it is based on properties of scientific publications in a particular domain, in this case publications of HEP. In the course of the refinement process, previously computed clusters can be completely discarded, extended with new elements, or some of their parts might be removed. In this subsection we discuss the heuristics applied for rejecting and splitting clusters of graphical operators.

There are two main reasons for rejecting a cluster. The first of them is a size being too small compared to a page size. The second is the figure candidate having its aspect ratio outside a desired interval of values.

The first heuristic is designed to remove small graphical elements appearing for example inside mathematical formulas, but also small logos and other decorations. The second one discards text separators and different parts of mathematical equations, such as a line-separating numerator from a denominator inside a fraction. The thresholds used for filtering are provided as configurable properties of the algorithm and their values are assigned experimentally in a way maximising the accuracy of figures detection.

Additionally, the analysis of the order of operations forming the content stream of a PDF document may help to split clusters that were incorrectly joined by Algorithm 1. Parts of the stream corresponding to logical parts of the document usually form a consistent subsequence. This observation allows the construction of a method of splitting elements incorrectly clustered together. We can assign content streams not only to entire PDF documents or pages, but also to every cluster of operations. The clustering algorithm presented in Algorithm 1 returns a set of areas with a list of operations assigned to each of them. The content stream of a cluster consists of all operations from such a set ordered in the same manner as in the original content stream of the PDF document. The usage of the original content stream allows us to define a distance in the content stream as follows:

> ***Definition 2.*** *If o_1 and o_2 are two operations appearing in the content stream of the PDF document, by the distance between these operations we understand the number of textual and graphical operations appearing after the first of them and before the second of them.*

To detect situations when a figure candidate contains unnecessary parts, the content stream of a figure candidate is read from the first to the last operation. For every two subsequent operations, the distance between them in the sense of the original content stream is calculated. If the value is larger than a given threshold, the content stream is split into two parts, which become separate figure candidates. For both candidates, a new boundary is calculated.

This heuristic is especially important in the case of less formal publications such as slides from presentations at conferences. Presentation slides tend to have a certain number of graphics appearing on every page and not carrying any meaning. Simple geometrical clustering would connect elements of page style with the rest of the document content. Measuring the distance in the content stream and defining a threshold on the distance facilitates the distinction between the layout and the rest of the page. This technique also might be useful to automatically extract the template used for a presentation, although this transcends the scope of this publication.

Clustering of Textual Operators

The same algorithm that clusters graphical elements can cluster parts of text. Detecting larger logically consistent parts of text is important because they should be treated as single entities during subsequent processing. This comprises, for example, inclusion inside a figure candidate (e.g., captions of axes, parts of a legend) and classification of a text paragraph as a figure caption.

Inclusion of Text Parts

The next step in figures extraction involves the inclusion of lost text parts inside figure candidates.

At the stage of operations clustering, only the operations of the same type (graphical or textual) were considered. The results of those initial steps become the input to the clustering algorithm that will detect relations between previously detected entities. By doing this, we move one level farther in the process of abstracting from operations. We start from basic meaningless operations. Later we detect parts of graphics and text, and finally we are able to see the relations between both.

Not all clusters detected at this stage are interesting because some might consist uniquely of text areas. Only those results that include at least one graphical cluster may be subsequently considered figure candidates.

Another round of heuristics marks unnecessary intermediate results as deleted. Applied methods are very similar to those described in "Filtering of Clusters" (above), only thresholds deciding on the rejections must change because we operate on geometrically much larger entities. Also the way of application is different—candidates rejected at this stage can be later restored to the status of a figure. Instead of permanently removing, heuristics of this stage only mark figure candidates as rejected. This happens in the case of the candidates having incorrect aspect ratio, incorrect sizes or consisting only of horizontal lines (which is usually the case with mathematical formulas but also tables).

In addition to using the aforementioned heuristics, having clusters consisting of a mixture of textual and graphical operations allows the application of new heuristics. During the next phase, we analyze the type of operations rather than their relative location. In some cases, steps described earlier might detect objects that should not be considered a figure, such as text surrounded by a frame. This situation can be recognized by the calculation of a ratio between the number of graphical and textual operations in the content stream of a figure candidate. In our approach we have defined a threshold that indicates which figure candidates should be rejected because they contain too few graphics. This allows the removal of, for instance, blocks of text decorated with graphics for aesthetic reasons. The ratio between numbers of graphical and textual operations is smaller for tables than for figures, so extending the heuristic with an additional threshold could improve the table–figure distinction. Another heuristic analyzes ratio between the total area of graphical operations and the area of the entire figure candidate.

Subsequently, we mark as deleted the figure candidates containing horizontal lines as the only graphical operations. These candidates describe tables or mathematical formulas that have survived previous steps of the algorithm. Tables can be reverted to the status of figure candidates in later stages of processing.

Figure candidates that survive all the phases of filtering are finally considered to be figures. Figure 2 shows a fragment of a publication page with indicated text areas and final figure candidates detected by the algorithm.

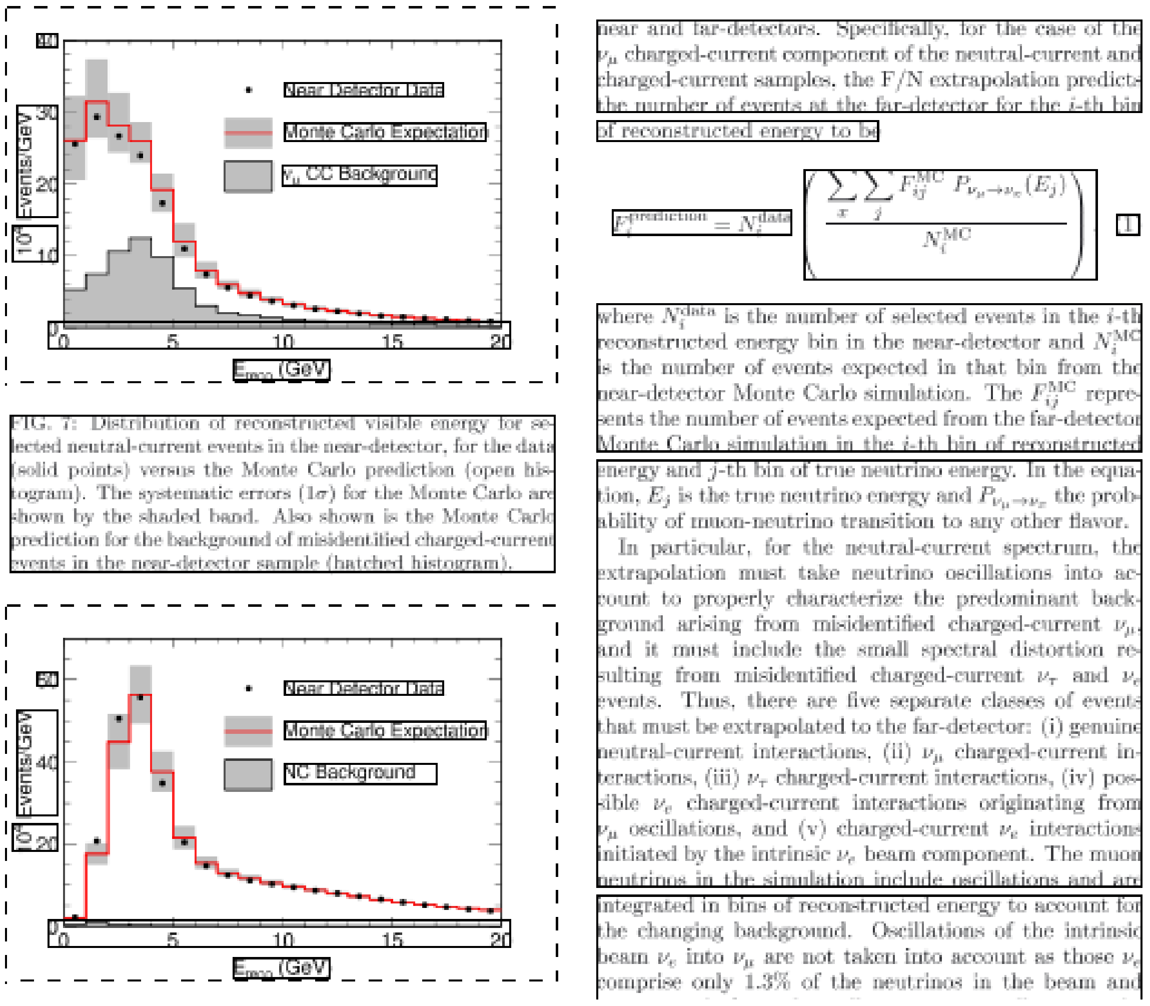


FIG. 7: Distribution of reconstructed visible energy for selected neutral-current events in the near-detector, for the data (solid points) versus the Monte Carlo prediction (open histogram). The systematic errors (1σ) for the Monte Carlo are shown by the shaded band. Also shown is the Monte Carlo prediction for the background of misidentified charged-current events in the near-detector sample (hatched histogram).

near and far-detectors. Specifically, for the case of the ν_μ charged-current component of the neutral-current and charged-current samples, the F/N extrapolation predicts the number of events at the far-detector for the i-th bin of reconstructed energy to be

$$F_i^{\text{prediction}} = N_i^{\text{data}} \left(\frac{\sum_x \sum_j F_{ij}^{\text{MC}} \, P_{\nu_\mu \to \nu_x}(E_j)}{N_i^{\text{MC}}} \right) \quad (1)$$

where N_i^{data} is the number of selected events in the i-th reconstructed energy bin in the near-detector and N_i^{MC} is the number of events expected in that bin from the near-detector Monte Carlo simulation. The F_{ij}^{MC} represents the number of events expected from the far-detector Monte Carlo simulation in the i-th bin of reconstructed energy and j-th bin of true neutrino energy. In the equation, E_j is the true neutrino energy and $P_{\nu_\mu \to \nu_x}$ the probability of muon-neutrino transition to any other flavor.

In particular, for the neutral-current spectrum, the extrapolation must take neutrino oscillations into account to properly characterize the predominant background arising from misidentified charged-current ν_μ, and it must include the small spectral distortion resulting from misidentified charged-current ν_τ and ν_e events. Thus, there are five separate classes of events that must be extrapolated to the far-detector: (i) genuine neutral-current interactions, (ii) ν_μ charged-current interactions, (iii) ν_τ charged-current interactions, (iv) possible ν_e charged-current interactions originating from ν_μ oscillations, and (v) charged-current ν_e interactions initiated by the intrinsic ν_e beam component. The muon neutrinos in the simulation include oscillations and are integrated in bins of reconstructed energy to account for the changing background. Oscillations of the intrinsic beam ν_e into ν_μ are not taken into account as those ν_e comprise only 1.3% of the neutrinos in the beam and

Figure 2. A fragment of the PDF page with boxes around every detected text area and each figure candidate. Dashed rectangles indicate figure candidates. Solid rectangles indicate text areas.

Detection and Matching of Captions

The input of the part of the algorithm responsible for detecting figure captions consists of previously determined figures and all text clusters. The observation of scientific publications shows that, typically, captions of figures start with a figure identifier (for instance see the grammar for figure captions proposed by Bathia, Lahiri, and Mitra.[22]

The identifier usually starts with a word describing a figure type and is followed by a number or some other unique identifier. In more complex documents, the figure number might have a hierarchical structure reflecting, for example, the chapter number. The set of possible figure types is very limited. In the case of HEP publications, the most usual combinations include words “figure”, “plot,” and different variations of their spelling and abbreviating.

During the first step of the caption detection, all text clusters from the publication page are tested for the possibility of being a caption. This consists of matching the beginning of the text contained in a textual cluster with a regular expression determining what is a figure caption. The role of the regular expression is to elect strings starting with one of the predefined words, followed by an identifier or beginning of a sentence. The identifier is subsequently extracted and included in the metadata of a caption. The caption detection has to be designed to reject paragraphs of the type "Figure 1 presents results of (. . .)". To achieve this, we reject the possibility of having any lowercase text after the figure identifier.

Having the set of all the captions, we start searching for corresponding figures. All previous steps of the algorithm take into account the division of a page into text columns (see "Detection of the Page Layout" below). When matching captions with figure candidates, we do not take into account the page layout.

Matching between figure candidates and captions happens at every document page separately. We consider every detected caption once, starting with those located at the top of the page and moving down toward the end. For every caption we search figure candidates lying nearby. First we search above the caption and, in the case of failure, we move below the caption. We take into account all figure candidates, including those rejected by heuristics.

In the case of finding multiple figure candidates corresponding to a caption, we merge them into a single figure, treating previous candidates as subfigures of a larger figure. We also include small portions of text and graphics previously rejected from figure candidates that lie between figure and caption and between different parts of a figure. These parts of text usually contain identifiers of the subfigures. The amount of unclustered content that can be included in a figure is a parameter of the extraction algorithm and is expressed as a percentage of the height of the document page.

It might happen that captions are located in a completely different location, but this case is rare and tends to appear in older publications. The distance from the figure is calculated based on the page geometry. The captions should not be too distant from the figure.

Generation of the Output

The choice of the format in which data should be saved at the output of the extraction process should take into account further requirements.

The most obvious use case of displaying figures to end users in response to text-based search queries does not yield very sophisticated constraints. A simple raster graphic annotated with captions and possibly some extracted portions of metadata would be sufficient. Unfortunately, the process of generating raster representations of figures might lose many important pieces of information that could be used in the future for an automatic analysis.

To store as much data as possible, apart from storing the extracted figures in a raster format (e.g., PNG), we also decided to preserve their original vector character. Vector graphics formats, similarly to PDF documents, contain information about graphical primitives. Primitives can be organized in larger logical entities. Sometimes rendering of different primitives leads to a modification of the same pixel of resulting image. Such a situation might happen, for example, when circles are used to draw data points lying nearby on the same plot. To avoid such issues, we convert figures into scalable vector graphics (SVG) format.[23]

On the implementation level, the extraction of vector representation of a figure proceeds in a manner similar to regular rendering of a PDF document. The interpreter preserves the same elements of the state and allows their modification by transformation operations. A virtual canvas is created for every detected figure. The content stream of the document is processed and all the transformation operations are executed modifying the interpreter's state. The textual and graphical operators are also interpreted, but they affect only the appropriate canvas of the figure to which the operation belongs. If a particular operation does not belong to any figure, no canvas is affected. The behaviour of graphical canvases used during the SVG generation is different from the case of raster rendering. Instead of creating graphical output, every operation is transformed into a corresponding primitive and saved within an SVG file.

PDF was designed in such a manner that the number of external dependencies of a file is minimized. This design decision led to the inclusion of the majority of fonts in the document itself. It would be possible to embed font glyphs in the SVG file and use them to render strings. However, for the sake of simplicity, we decided to omit font definitions in the SVG output.

A text representation is extracted from every text operation, and the operation is replaced by a SVG text primitive with a standard font value. This simplification affects what the output looks like, but the amount of formatting information that is lost is minimal. Moreover, this does not pose a problem because vector representations are intended to be used during automatic analysis of figures rather than for displaying purposes. A possible extension of the presented method could involve embedding complete information about used glyphs.

Finally, the generation of the output is completed with some metadata elements. An exhaustive categorization of the metadata that can be compiled for figures could be the customization of the one proposed by Liu et al. for table metadata.[24] In the case of figures, the following categories could be distinguished: (1) environment/geography metadata (information of the document where the figure is located); (2) affiliated metadata (e.g., captions, references, or footnotes); (3) layout metadata (information about the original visualization of the figure); (4) content data; and (5) figure type metadata. For the moment, we compile only environment/geography metadata and affiliated metadata.

The geography/environment metadata consists of the document title, the document authors, the document date (creation and publication), and the exact location of a figure inside a publication (page and boundary). Most of these elements are provided by simply referencing the original publication in the INSPIRE repository. The affiliated metadata consists of the text caption and the exact location of the caption in the publication (page and boundary). In the future, metadata from other categories will be annotated for each figure.

Detection of the Page Layout

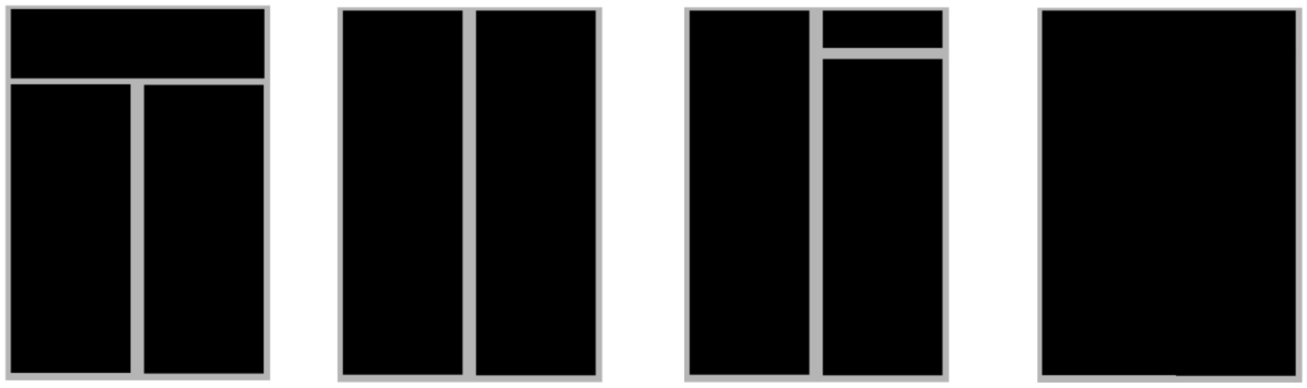


Figure 3. Sample page layouts that might appear in a scientific publication. The black color indicates areas where content is present.

In this section we discuss how to detect the page layout, an issue which has been omitted in the main description of the extraction algorithm, but which is essential for an efficient detection of figures. Figure 3 depicts several possibilities of organising content on the page. As mentioned in previous sections, the method of clustering operations based on their geometrical position may fail in the case of documents having a complex page layout. The content appearing in different columns should never be considered belonging to the same figure. This cannot be assured without enforcing additional constrains during the clustering phase.

To address this difficulty, we enhanced the figure extractor with a pre-processing phase of detecting the page layout. Being able to identify how the document page is divided into columns enables us to execute the clustering within every column separately. It is intuitively obvious, what can be understood as a page layout, although to provide a method of calculating such, we need a more formal definition, which we provide below.

By the layout of a page, we understand a particular division of a page into areas called columns. Each area is a sum of disjoint rectangles. The division of a page into areas must satisfy a set of conditions summarized in definition 3.

__Definition 3:__ Let P be a rectangle representing the page. The set D containing subareas of a page is called a page division if and only if

$$\bigcup_{Q\in D} Q = P$$
$$\forall_{x,y\in D}\, x \cap y = \emptyset$$
$$\forall_{Q\in D}\, Q \neq \emptyset$$
$$\forall_{Q\in D}\, \exists_{R=\{x:x\ is\ a\ rectangle, \forall_{y\in R\setminus\{x\}}\, y\cap x=\emptyset\}}\, Q = \bigcup_{x\in R} x$$

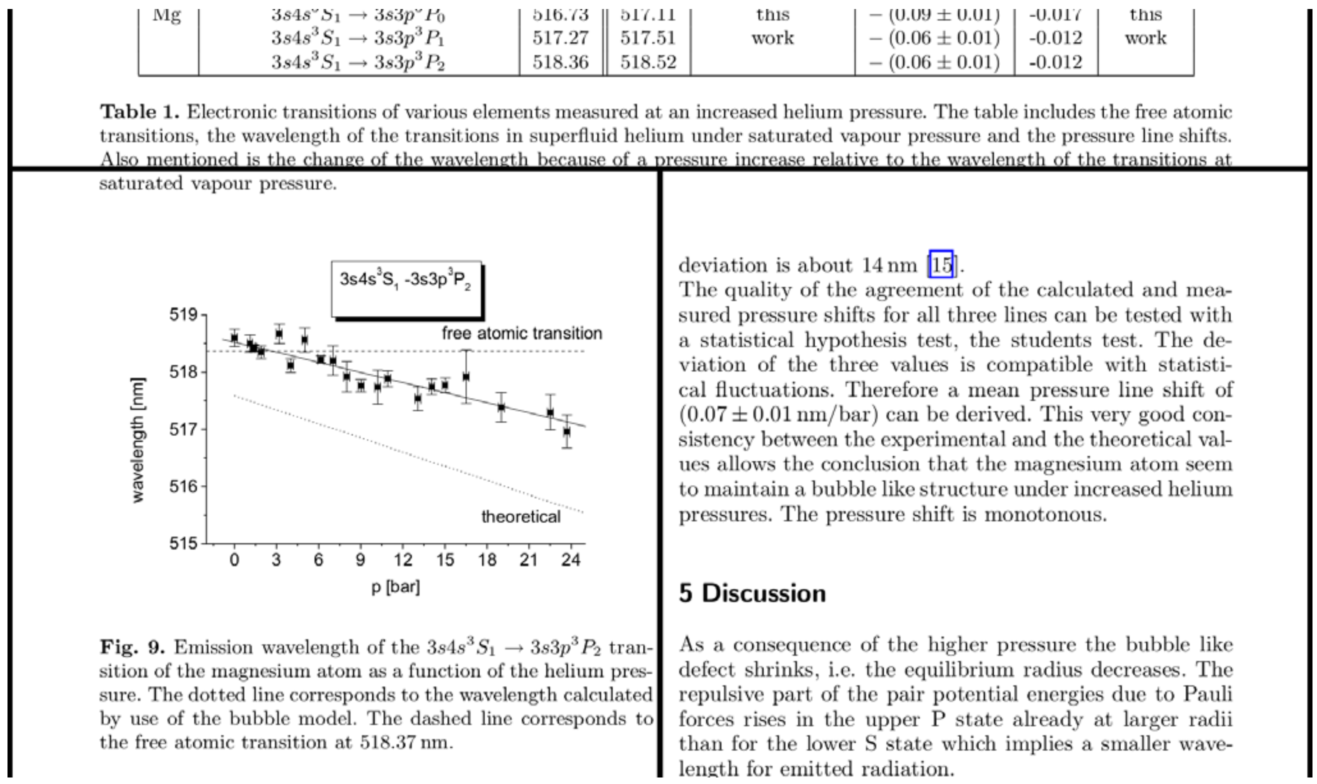
Every element of a division is called a page area.

To be considered a page layout, borders of areas from the division must not intersect the content of the page.

Definition 3 does not guarantee that the layout is unique. A single page might be assigned different divisions satisfying the definition. Additionally, not all valid page layouts are interesting from the point of view of figures detection. The segmentation algorithm calculates one of such divisions, imposing additional constraints on the detected areas. The layout-calculation procedure utilizes the notion of separators, introduced by definition 4.

__Definition 4:__ A vertical (or horizontal) line inside a page or on its borders is called a separator if its horizontal (vertical) distance from the page content is larger than a given constant value.

The algorithm consists of two stages. First, the vertical separators of a sufficient length are detected and used to divide the page into disjoint rectangular areas. Each area is delimited by two vertical lines, each of which forms a consistent interval inside of one of the detected vertical separators. At this stage, horizontal separators are completely ignored. Figure 4 shows a fragment of a publication page processed by the first stage of the layout-detection. The upper horizontal edge of one of the areas lies too close too close to two text lines. With the constant of the definition 4 chosen to be sufficiently large, this edge would not be a horizontal separator and thus the generated division of the page would require additional processing to become a valid page layout. The second stage of the algorithm transforms the previously detected rectangles into a valid page layout by splitting rectangles into smaller parts and by joining appropriate rectangles to form a single area.



Mg	$3s4s^3S_1 \rightarrow 3s3p^3P_0$	516.73	517.11	this	$-(0.09 \pm 0.01)$	-0.017	this
	$3s4s^3S_1 \rightarrow 3s3p^3P_1$	517.27	517.51	work	$-(0.06 \pm 0.01)$	-0.012	work
	$3s4s^3S_1 \rightarrow 3s3p^3P_2$	518.36	518.52		$-(0.06 \pm 0.01)$	-0.012	

Table 1. Electronic transitions of various elements measured at an increased helium pressure. The table includes the free atomic transitions, the wavelength of the transitions in superfluid helium under saturated vapour pressure and the pressure line shifts. Also mentioned is the change of the wavelength because of a pressure increase relative to the wavelength of the transitions at saturated vapour pressure.

Fig. 9. Emission wavelength of the $3s4s^3S_1 \rightarrow 3s3p^3P_2$ transition of the magnesium atom as a function of the helium pressure. The dotted line corresponds to the wavelength calculated by use of the bubble model. The dashed line corresponds to the free atomic transition at 518.37 nm.

deviation is about 14 nm [15].
The quality of the agreement of the calculated and measured pressure shifts for all three lines can be tested with a statistical hypothesis test, the students test. The deviation of the three values is compatible with statistical fluctuations. Therefore a mean pressure line shift of $(0.07 \pm 0.01$ nm/bar) can be derived. This very good consistency between the experimental and the theoretical values allows the conclusion that the magnesium atom seem to maintain a bubble like structure under increased helium pressures. The pressure shift is monotonous.

5 Discussion

As a consequence of the higher pressure the bubble like defect shrinks, i.e. the equilibrium radius decreases. The repulsive part of the pair potential energies due to Pauli forces rises in the upper P state already at larger radii than for the lower S state which implies a smaller wavelength for emitted radiation.

Figure 4. Example of intermediate layout-detection results requiring the refinement.

Algorithm 2 shows the pseudo-code of the detection of vertical separators. The input of the algorithm consists of the image of the publication page. The output is a list of vertical separators aggregated by their x-coordinates. Every element of this list consists of two elements: an integer indicating the x-coordinate and the list of y-coordinates describing the separators. The first element of this list indicates the y-coordinate of the beginning of the first separator. The second element is the y-coordinate of the end of the same separator. The third and fourth elements describe the second separator and the same mechanism is used for the remaining separators (if they exist).

The algorithm proceeds according to the sweeping principle known from the computational geometry.[25] The algorithm reads the publication page starting from the left. For every x-coordinate value, a set of corresponding vertical separators is detected (lines 9–18). Vertical separators are searched as consistent sequences of blank points. A point is considered blank if all the points in its horizontal surrounding of the radius defined by the constant from definition 5 are of the background colour. Not all blank vertical lines can be considered separators. Short, empty spaces usually delimit lines of text or different small units of the content. In line 11 we test detected vertical separators for being long enough.

If a separator has been detected in a particular column of a publication page, the adjacent columns also tend to contain similar separators. Lines 19–31 of the algorithm are responsible for electing the longest candidate among the adjacent columns of the page. The maximization is performed across a set of adjacent columns for which at least one separator exists.

```
1:  Input: the page image
2:  Output: vertical separators of the input page
3:  List<Pair<int, List<int>> separators ← ∅
4:  int max_weight ← 0;
5:  boolean maximizing ← false
6:  for all x ∈ {$min_x$ … $max_x$} do
7:      $empty_b$ ← 0, current_eval ← 0
8:      empty_areas ← List()
9:      for all y ∈ {0 … page_height} do
10:         if point at (x, y) is not blank then
11:             if $y - empty_b - 1 > height_{min}$ then
12:                 empty_areas.append($empty_b$)
13:                 empty_areas.append(y = page_height? y : y-1)
14:                 current_eval ← current_eval + y - $empty_b$
15:             end if
16:             $empty_b$ ← y + 1
17:         end if
18:     end for
    {We have already processed the entire column. Now we are comparing with adjacent
    already processed columns}
19:     if max_weight < current_eval then
20:         max_weight ← current_eval
21:         max_separators ← empty_areas
22:         $max_x$ ←  x
23:     end if
24:     if maximising then
25:         if empty_areas = ∅ then
26:             separators.add(<$max_x$, max_separators>)
27:             maximising ← false, max_weight ← 0
28:         end if
29:     else
30:         maximising ← (empty_areas ≠ ∅)
31:     end if
32: end for
33: return separators
```

Algorithm 2. Detecting vertical separators.

The detected separators are used to create the preliminary division of the page, similar to the one from the example of figure 4. As with the previous step, separators are considered one by one in the order of increasing x coordinate. At every moment of the execution, the algorithm maintains a division of the page into rectangles. This division corresponds only to the already detected vertical separators. Updating the previously considered division is facilitated by processing separators in a particular well-defined order.

Before presenting the final outcome, the algorithm must refine the previously calculated division. This happens in the second phase of the execution. All the horizontal borders of the division are then moved along adjacent vertical separators until they become horizontal separators in the sense of definition 4. Typically, moving the horizontal borders result in dividing already existing rectangles into smaller ones. If such a situation happens, both newly created parts are assigned to different page layout areas. Sometimes when moving separators is not possible, different areas are combined together, forming a larger one.

Tuning and Testing

The extraction algorithm described here has been implemented in Java and tested on a random set of scientific articles coming from the Inspire repository. The testing procedure has been used to evaluate the quality of the method, but also allowed to tweak the parameters of the algorithm to maximize the outcomes.

Preparation of the Testing Set

To prepare the testing set, we randomly selected 207 documents stored in INSPIRE. In total, these documents consisted of 37,28 pages which contained 1,697 figures altogether.

The records have been selected according to a uniform probability distribution across the entire record space. This way, we have created a collection that is representative for the entire INSPIRE including historical entries.

Currently, INSPIRE consists of: 1,140 records describing publications written before 1950; 4,695 between 1950 and 1960; 32,379 between 1960 and 1970; 108,525 between 1970 and 1980; 167,240 between 1980 and 1990; 251,133 between 1990 and 2000; and 333,864 in the first decade of the twenty-first century. In total, up to July 2012, INSPIRE manages 952,026 records. It can be seen that the rate of growth has increased with time and most of INSPIRE documents come from the last decade.

The results on such a testing set should accurately estimate the efficiency of extraction for existing documents but not necessarily for new documents, being ingested into INSPIRE. This is because INSPIRE contains entries describing old articles which were created using obsolete technologies or scanned and encoded in PDF. The extraction algorithm is optimized for born-digital objects. To test the hypothesis that the extractors provides better results for newer papers, the testing set has been split into several subsets. The first set consists of publications published before 1980. The rest of the testing set has been split into subsets corresponding to decades of publication.

To simplify the counting of correct figure detections and to provide a more reliable execution and measurement environment, every testing document has been split into many of PDF documents consisting of a single page. Subsequently, every single page document has been manually annotated with the number of figures appearing inside.

Execution of the Tests

The efficient execution of the testing was possible thanks to a special script executing the plots extractor on every single page separately and then computing the total number of successes and failures. The script allows the execution of tests in a distributed heterogeneous environment and allows dynamic connection and disconnection of computing nodes. In the case of a software failure, the extraction request is resubmitted to a different computation node, allowing the avoidance problems related to a worker node configuration rather than to the algorithm implementation itself.

During the preparation of the testing set, we manually annotated all the expected extraction results. Subsequently, the script compared these metadata with the output of the extractor. Using aggregated numbers from all extracted pages allowed us to calculate efficiency measures of the extraction algorithm. As quality measures, we used recall and precision.[26] Their definitions are included in the following equations:

$$recall = \frac{\#\, correctly\ extracted\ \ figures}{\#\ figures\ \ present\ in\ the\ testset}$$

$$precision = \frac{\#\, correctly\ extracted\ \ figures}{\#\, extracted\ \ figures}$$

At every place where we needed a single comparable quality measure rather than two semi-independent numbers, we have used a harmonic average of the precision and the recall.[27]

$$\boldsymbol{harmonic\ average} = \frac{\mathbf{2}}{\dfrac{\mathbf{1}}{\boldsymbol{precision}} + \dfrac{\mathbf{1}}{\boldsymbol{recall}}}$$

Table 1 summarizes the results obtained during the test execution for every subset of our testing set. Figure 5 shows the dependency of recall and precision on the time of publication. The extractor parameters used in this test execution were chosen based on intuition and small number of manually triggered trials. In the next section we describe an automatic tuning procedure we have used to find the most optimal algorithm arguments.

	–1980	1980–90	1990–2000	2000–10	2010–12
Number of existent figures	114	60	170	783	570
Number of correctly detected figures	59	53	164	703	489
Number of incorrectly detected figures	26	78	65	40	73
Total number of pages	85	136	760	1919	828
Number of correctly processed pages	20	44	712	1816	743

Table 1. Results of the test execution.

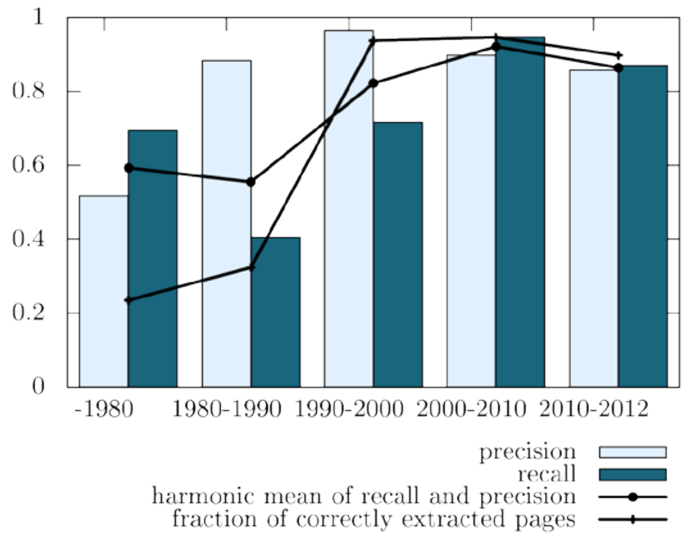


Figure 5. Recall and precision as functions of decade of the date of the publication.

It can be seen that, as expected, the efficiency increases with the increasing time of publication. A total recall and precision for all samples since 1990, which constitutes a majority of the INSPIRE corpus, were both 88 percent.

Precision and recall based on the correctly detected figures do not give a full image of the algorithm efficiency because the extraction has been executed on a number of pages not containing any figures. The correctly extracted pages not having any figures do not appear in the recall and precision statistics because in their case the expected and detected number of figures are both equal to 0.

Besides recall and precision, figure 5 depicts also the fraction of pages that have been extracted correctly. Taking into account the samples since 1990, 3,271 pages out of 3,507 have been detected completely correctly, which makes 93 percent success rate counted by number of pages. As it can be seen, this measure is higher than both the precision and the recall.

The analysis of the extractor results in the case of failure shows that in many cases, even if results are not completely correct, they are not far from the expectation. There are different reasons of the algorithm failing. Some of them may result from non-optimal choice of algorithm parameters, others from document layout being too far from the assumed one. In some rare cases, even manual inspection of the document does not allow an obvious identification of figures.

The Automatic Tuning of Parameters

In previous section we have shown the results obtained by executing the extraction algorithm on a sample set. During this execution we were using extractor arguments which seemed to be the most correct based on our observation but also on other research (typical sizes of figures, margin sizes, etc.).[28] This way of algorithm configuration was useful during the development, but is not likely to yield the best possible results. To find better parameters, we have implemented a method of automatic tuning. Metrics described in the previous section provided a good method of measuring the efficiency of the algorithm running based on given parameters.

The choice of optimal parameters can be relative to the choice of documents on which the extraction is to be performed. The way in which the testing set has been selected, allowed us to use it as representative for the HEP publications. To tune the algorithm, we have used a described subset of testing set from the previous step as a reference. The subset consisted of all entries created after 1990. This allowed us to minimize the presence of scanned documents which, by design, cannot be correctly processed by our method.

The adjustment of parameters has been performed by a dedicated script which has executed the extraction using various parameter values and has read results. The script has been configured with a list of tuneable parameters together with their type and allowed values range. Additionally, the script had the knowledge of the believed best value, which was the one used in previous testing.

To decrease the complexity of training, we have made several assumptions about the parameters. These assumptions are only an approximation of real nature of parameters, but the practice has shown that they are good enough to permit the optimization:

- We assume that the precision and recall are continuous with respect to the parameters. This allows us to assume that efficiency of the algorithm for parameter values close to a given one will be close. The optimization has proceeded by sampling the parametric space in a number of points and executing tests using the selected points as parameter values.

Having N parameters to optimize and dividing the space of every parameter into M regions leads to the execution of M^N tests. Execution of every test is a timely operation due to the size of the training set.

- We assume that parameters are independent from each other. This means that we can divide the problem of finding an optimal solution in the N-dimensional space of N configuration arguments into finding N solutions in 1-dimensional subspaces. Such an assumption seems to be intuitive and considerably reduces the number of necessary tests from $O(M^N)$ to $O(M \cdot N)$, where M is the number of samples taken from a single dimension.

In our tests, the parametric space has been divided into 10 equal intervals in every direction. In addition to checking the extraction quality in those points, we have executed one test for the so-far best argument. In order to increase the level of fine-tuning of the algorithm, each test has been re-executed in the region, where chances of finding a good solution were considered the highest. This consisted of a region centred around the highest result and having a radius of 10 percent of the parameter space.

Figure 6 and figure 7 show the dependency of the recall and the precision on an algorithm parameter. The parameter depicted in figure 6 indicates what minimal aspect ratio the figure candidate must have in order to be considered a correct figure. It can be seen that tuning this heuristic increases the efficiency of the extraction. Moreover, the dependency of recall and precision on the parameter is monotonic which is the most compatible with the chosen optimization method.

The parameter of figure 7 specifies which fraction of the area of the entire figure candidate has to be occupied by graphical operations. This parameter has a lower influence on the extraction efficiency. Such a situation can happen when more than one heuristic influences the same aspect of the. This is contradictory with the assumption of parameter independence, but we have decided to use the present model for the simplicity.

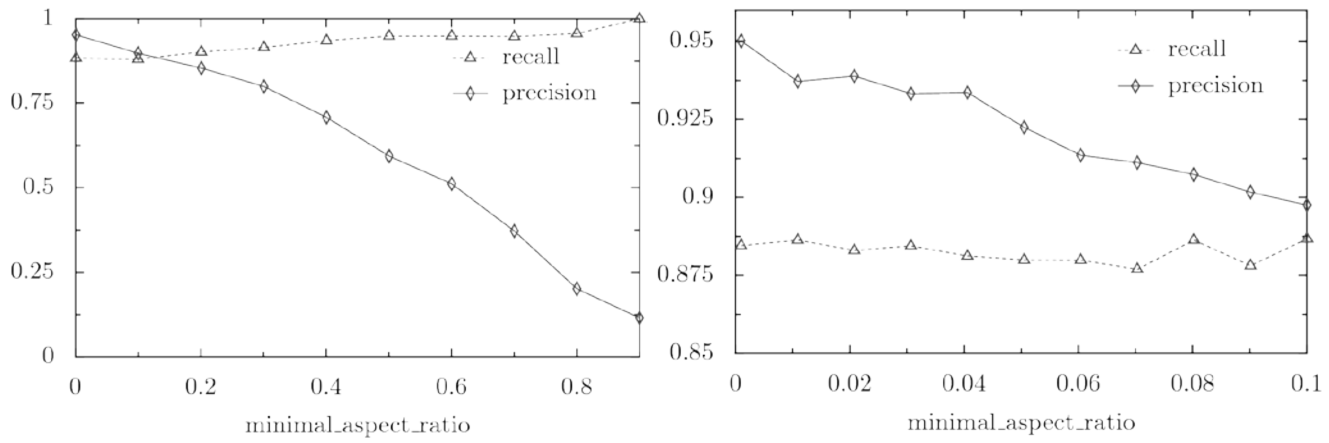


Figure 6. Effect of the minimal aspect ratio on precision and recall.

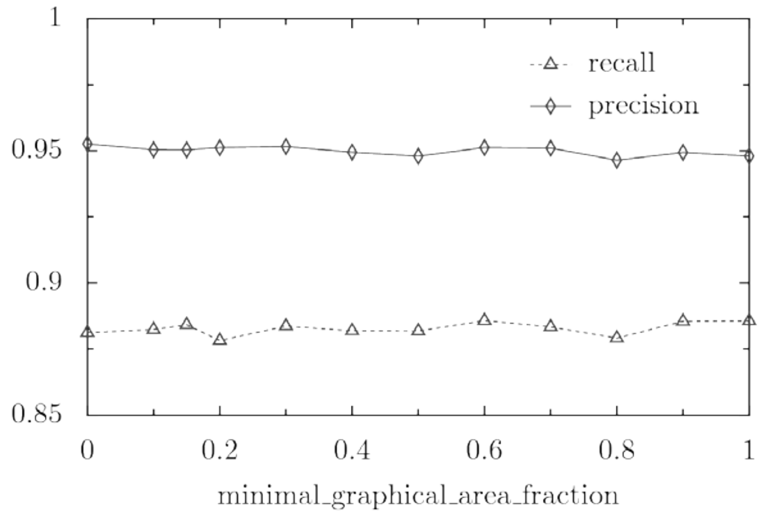


Figure 7. Effect on the precision and recall of the area fraction occupied by graphical operations.

After executing the optimization algorithm, we have managed to achieve a recall of 94.11 percent and a precision of 96.6 percent, which is a considerable improvement compared to previous results of 88 percent.

CONCLUSIONS AND FUTURE WORK

This work has presented a method for extracting figures from scientific publications in a machine-readable format, which is the main step toward the development of services enabling access and search of images stored in scientific digital libraries. In recent years, figures have been gaining increasing attention in the digital libraries community. However, little has been done to decipher the semantics of these graphical representations and to bridge the semantic gap between content, which can be understood by machines and this which is managed by digital libraries. Extracting figures and storing them in uniform and machine-readable format constitutes the first step towards the extraction and the description of the internal semantics of figures. Storing semantically described and indexed figures would open completely new possibilities of accessing the data and discovering connections between different types of publishing artefacts and different resources describing related knowledge.[29]

Our method of detecting fragments of PDF documents that correspond to figures is based on a series of observations of the character of publications. However, tests have shown that additional work is needed to improve the correctness of the detection. Also, the performance should be re-evaluated after we have a large set of correctly annotated figures, confirmed by users of our system. The heuristics used by the algorithm are based on a number of numeric parameters that we have tried to optimize using automatic techniques. The tuning procedure has made several arbitrary assumptions on the nature of the dependency between parameters and extraction results. A future approach to the parameter optimization, requiring much more processing, could involve the execution of a genetic algorithm that would treat the parameters as gene samples.[30] This could potentially allow a discovery of a better parameter set because a smaller set of assumptions would be imposed on the parameters. A vector of algorithm parameters could play the role of a gene and random mutations could be introduced to previously considered and subsequently crossed genes. The evaluation and selection of surviving genes could be performed by the usage of the metrics described previously. Another approach to improving the quality of the tuning could involve extending the present algorithm by a discovery of mutually dependent parameters and usage of special techniques (relaxing the assumptions) to fine-tune in subspaces spanned by these parameters.

All of our experiments have been performed using a corpus of publications from HEP. The usage of the extraction algorithm on a different corpus would require tuning the parameters for the specific domain of application. For the area of HEP, we can also consider preparing several sets of execution parameters varying by decade of document publication or by other easy to determine characteristics. Subsequently, we could decide which extraction method to run, based on those metrics.

In addition to a better tuning of the existing heuristics, there are improvements that can be made at the level of the algorithm. For example, we could mention extending the process of clustering text parts. In the current implementation, the margins by which textual operations are extended during the clustering process are fixed as algorithm parameters. This approach proved to be robust in most cases. In fact, distances between text lines tend to be different depending on the currently utilized style. Every text portion tends to have one style that dominates. An improved version of the text-clustering algorithm could use local rather than global properties of the content. This would not only allow to correctly handle the entire document written using different text styles, but also help to manage cases of single paragraphs differing from the rest of the content.

Another important, not-yet-implemented improvement related to figure metadata is the automatic extraction of figure references from the text content. Important information about figure content might be stored in the surroundings of the place where publication text refers to a figure. Furthermore, the metadata could be extended by the usage of some type of classifier that would assign a graphics type to the extracted result. Currently, we are only distinguishing between tables and figures based on simple heuristics involving number and type of graphical areas and the text inside of the detected caption. In the future, we could detect line-plots from photos, histograms and so on. Such a classifier could be implemented using artificial intelligence techniques such as support vector machines.[31]

Finally, partial results of the figures extraction algorithm might be useful in performing other PDF analyses:

- The usage of clustered text areas could allow a better interpretation and indexing of textual content stored in digital libraries with full-text access. Clusters of text tend to describe logical parts like paragraphs, section and chapter titles, etc. A simple extension of the current schema could allow the extraction of predominant formatting style of the text encoded in a page area. Text parts written in different styles could be indexed in a different manner giving for instance more importance to segments written with larger font.

- We mentioned that the algorithm detects not only figures, but also tables. A heuristic is being used in order to distinguish tables from different types of figures. Our present effort concentrates on correct treatment of figures, but a useful extension could allow extraction of different types of entities. For instance, another common type of content ubiquitous in HEP documents are mathematical formulas. Thus, in addition to figures, it would be important to extract tables and formulas in structured format allowing a further processing.

The internal architecture of the implemented prototype of the figure extractor allows easy implementation of extension modules which can compute other properties of PDF documents.

ACKNOWLEDGEMENTS

This work has been partially supported by CERN, and the Spanish Government through the project TIN2012-37826-C02-01.

REFERENCES

1. Saurabh Kataria, "On Utilization of Information Extracted From Graph Images in Digital Documents," *Bulletin of IEEE Technical Comittee on Digital Libraries* 4, no. 2 (2008), http://www.ieee-tcdl.org/Bulletin/v4n2/kataria/kataria.html.
2. Marti A. Hearst et al., "Exploring the Efficacy of Caption Search for Bioscene Journal Search interfaces," *Proceedings of the Workshop on Bio NLP 2007: Biological, Translational and Clinical Language Processing*: 73–80, http://dl.acm.org/citation.cfm?id=1572406.
3. Lisa Johnston, "Web Reviews: See the Science: Scitech Image Databases," *Sci-Tech News* 65, no. 3 (2011), http://jdc.jefferson.edu/scitechnews/vol65/iss3/11.
4. Annette Holtkamp et al., "INSPIRE: Realizing the Dream of a Global Digital Library in High-Energy Physics," *3rd Workshop Conference: Towards a Digital Mathematics Library*, Paris, France (July 2010): 83–92.
5. Piotr Praczyk et al., "Integrating Scholarly Publications and Research Data—Preparing for Open Science, a Case Study from High-Energy Physics with Special Emphasis on (Meta)data Models," *Metadata and Semantics Research—CCIS* 343 (2012): 146–57.
6. Piotr Praczyk et al., "A Storage Model for Supporting Figures and Other Artefacts in Scientific Libraries: the Case Study of Invenio," *4th Workshop on Very Large Digital Libraries (VLDL 2011)*, Berlin, Germany (2011).

7. "SciVerse Science Direct: Image Search," Elsevier, http://www.info.sciverse.com/sciencedirect/using/searching-linking/image.

8. Guenther Eichhorn, "Trends in Scientific Publishing at Springer," in *Future Professional Communication in Astronomy II* (New York: Springer, 2011), doi: 10.1007/978-1-4419-8369-5_5.

9. William Browuer et al., "Segregating and Extracting Overlapping Data Points in Two-dimensional Plots," *Proceedings of the 8th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2008)*, New York: 276–79.

10. Saurabh Kataria et al., "Automatic Extraction of Data Points and Text Blocks from 2-Dimensional Plots in Digital Documents," *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, (2008) Chicago: 1169–1174.

11. Saurabh Kataria, "On Utilization of Information Extracted From Graph Images in Digital Documents," *Bulletin of IEEE Technical Committee on Digital Libraries* 4, no. 2 (2008), http://www.ieee-tcdl.org/Bulletin/v4n2/kataria/kataria.html.

12. Ying Liu et al., "Tableseer: Automatic Table Metadata Extraction and Searching in Digital Libraries," *Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'07)*, Vancouver (2007): 91–100.

13. William S. Cleveland, "Graphs in Scientific Publications," *American Statistician*, 38, no. 4, (1984): 261–69, doi: 10.1080/00031305.1984.10483223.

14. Hui Chao and Jian Fan, "Layout and Content Extraction for PDF Documents," *Document Analysis Systems VI*, *Lecture Notes in Computer Science* 3163 (2004): 213–24.

15. At every moment of the execution of a PostScript program, the interpreter maintains many variables. Some of them encode current positions within the rendering canvas. Such positions are used to locate the subsequent character or to define the starting point of the subsequent graphical primitive.

16. Transformation matrices are encoded inside the interpreters' state. If an operator requires arguments indicating coordinates, these matrices are used to translate the provided coordinates to the coordinate system of the canvas.

17. Graphical operators are those that trigger the rendering of a graphical primitive.

18. Textual operations are the PDF instructions that cause the rendering of the text. Textual operations receive the string representation of the desired text and use the current font, which is saved in the interpreters' state.

19. Operations that do not produce any visible output, but solely modify the interpreters' state.

20. Herbert Edelsbrunner and Hermann A. Maurer, "On the Intersection of Orthogonal Objects," *Information Processing Letters* 13, nos. 4, 5 (1981): 177–81.

21. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, (Cambridge: MIT Electrical Engineering and Computer Science Series, 1990).

22. Sumit Bhatia, Shibamouli Lahiri, and Prasenjit Mitra, "Generating Synopses for Document-Element Search," *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, New York (2009): 2003–6, doi: 10.1145/1645953.1646287.

23. Jon Ferraiolo, ed., "Scalable Vector Graphics (SVG) 1.0 Specification," *W3C Recommendation 01 September 2001*, http://www.w3.org/TR/SVG10/.

24. Liu et al., "Tableseer."

25. Cormen, Leiserson, and Rivest, *Introduction to Algorithms*.

26. Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto, *Modern Information Retrieval*," (Boston: Addison-Wesley, 1999).

27. Ibid.

28. Cleveland, "Graphs in Scientific Publications."

29. Praczyk et al., "A Storage Model for Supporting Figures and Other Artefacts in Scientific Libraries."

30. Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach (Third Edition)* (Prentice Hall, 2009).

31. Sergios Theodoridis and Konstantinos Koutroumbas, *Pattern Recognition (Third Edition)* (Boston, Academic Press, 2006).