J Braz Comput Soc (2012) 18:331–349
DOI 10.1007/s13173-012-0066-6

ORIGINAL PAPER

# Optimal pagination and content mapping for customized magazines

**Ricardo Piccoli · João Oliveira · Isabel Manssour**

Received: 20 July 2011 / Accepted: 17 February 2012 / Published online: 14 March 2012


**Abstract** Traditional media such as magazines and newspapers are undergoing deep transformations as they cope with the high volume and dynamicity of currently available information. In addition, with the emergence of decentralized publishing models, there is an increasing need for automated tools for authoring high-quality documents. Moreover, much of the dynamic information on the Web could also profit from such mechanisms for automatic presentation and summarization.

This paper describes a solution to the problem of automatically producing a camera-ready magazine from a set of page templates and a sequence of variable content to be placed on those templates. The algorithm is able to find the optimal number of pages to hold the content, selecting the best templates to be used in the magazine in such a way that all pages are optimally used.

The algorithm was integrated to Adobe's InDesign® software, extending it to perform text fitting and rendering of magazine pages. The complete workflow is described in this paper, as well as an empirical evaluation and a discussion of future research directions.

**Keywords** Variable data printing · Automatic document layout · Mass customization · Customized magazines

R. Piccoli (✉) · J. Oliveira · I. Manssour
Centro de Pesquisa em Computação Aplicada, Faculdade de Informática, PUCRS, Porto Alegre, Brazil
e-mail: rfbpiccoli@gmail.com

J. Oliveira
e-mail: oliveira@inf.pucrs.br

I. Manssour
e-mail: manssour@inf.pucrs.br

## 1 Introduction

Information is currently produced, consumed and delivered in ways that directly affect traditional publishing businesses such as magazine publishers, where information (e.g. news stories, etc.) is usually produced by journalists and placed on pages designed to hold information with sizes known in advance. Such a scenario is changing as centralized publishing models are unable to cope with the high volume and dynamicity of information available from a myriad of sources. Therefore, a new model for publishing is required, combining graphical qualities and dynamic content obtained from the web.

Another noticeable trend is the appearance of decentralized self-publishing businesses, where any individual is able to create and distribute their own publications. An example of this trend is HP's MagCloud® service: clients are able to send their own PDF [5] files and the service takes care of displaying, charging, printing and distributing.[1]

However, the task of assembling graphics and content to produce one's own magazine usually requires graphic design skills and technical knowledge of a desktop publishing tool such as Adobe InDesign®[2] or QuarkXPress®[3] before a PDF file is produced. In this case, an automated system for assembling arbitrary content into camera-ready publications is still a desirable technology.

Systems for assembling documents are not new. Variable Data Printing (VDP) [4] systems evolved from earlier transactional businesses such as direct mail marketing, bank statements, bills and others [9], where static content is mixed

[1] http://magcloud.com.

[2] http://www.adobe.com/products/indesign.

[3] http://www.quark.com.

with custom data (i.e. relevant client information). The goal was to assemble large numbers of document instances that are unique and targeted at single individuals. However, most VDP-related technologies, such as the PPML language [1] can only handle content that is more or less predictable, and a graphic artist must prepare for little document variability in order to produce a document template that serves as base for the customized instances.

### 1.1 Objectives and contributions of this paper

This work describes a method for the automatic construction of magazines where an arbitrary sequence of content such as text and pictures is optimally mapped to designer-generated templates, ensuring the best possible presentation of the content. In addition, users may require the content to fill a specified number of pages, or otherwise let the system automatically determine an optimal number for that content. Thus, this problem can be reduced to mapping content into one or more pages and selecting appropriate templates for each page, maximizing some quality measure. This is known in the literature as the pagination problem.

The proposed algorithm receives a content sequence and produces an optimal sequence of template selections matching the content. After producing the optimal content mapping and pagination, Adobe's InDesign® software is used as a rendering engine. An extension to InDesign imports the layouts and performs typographic corrections such as text fitting on templates' text placeholders.

In addition to the algorithm for mapping content to templates, other contributions of this paper are:

- A simple and extensible representation for different types of content;
- A complementary solution to the recent works of Giannetti [9], to handle both pagination and text flow issues;
- The algorithm is able to fit the content to a specific number of pages provided by the user, or select an optimal number by itself.
- Apart from other pagination methods, the proposed pagination algorithm always fills every page completely, including the last page.
- It is very easy for a user to perform changes in content or templates and quickly produce a new document for proofreading.

This paper is organized as follows. Section 2 describes related works. Next, in Sect. 3 the problem is described and the requirements for an optimal pagination are defined. The workflow for the solution is described in the same section and document and template representations are detailed in Sect. 4. The mapping and pagination algorithms are described in Sect. 5 and the generation of camera-ready documents is described in Sect. 6. In Sect. 7 results and performance are presented. Section 8 describes additional techniques for enhancing results through the use of parameters used by the algorithm. Finally, Sect. 9 present conclusions, and a discussion on possible improvements and future directions of this work.

## 2 Related works

Purely automated solutions to the construction of documents have been studied for years as the automatic page layout problem [11, 13, 15, 16, 19, 21].

It is usually hard to produce an aesthetically pleasing layout starting from unknown, variable content. To ensure aesthetic quality and reduce the search space, current approaches may constrain the problem to less flexible layouts and well-behaved content. Newspapers are an example where the layout is hierarchical by nature and news items are similar in appearance and length, rendering the construction of deterministic algorithms for this task less hard [19].

It is worth noting that given the complexity of the layout problem, several non-deterministic approaches for automatic document layout have been proposed [11, 15, 21]. These approaches attempt to maximize an objective function that encodes one or more aesthetic qualities, and rely on randomized optimization heuristics, such as simulated annealing [8] or genetic algorithms [10] to find the best layout. However, these approaches are known to require long execution times in order to produce acceptable quality documents, and usually present poor convergence to optimal solutions when given (often conflicting) multiple aesthetic criteria [12, 21]. Given the predictability of results, deterministic approaches are thus more desirable in a document production setting, where documents cannot be manually inspected for aesthetic flaws.

For most high-end publications such as magazines, a graphic designer produces page templates to be used in the publication, conveying the look-and-feel and appeal of that particular magazine. Approaches for the selection of templates based on grids have been investigated by Jacobs et al. [14, 22], describing a document authoring system using constraint-based templates to have multiple geometric representations for the same template depending on its content. Their aim was to enable a designer to provide multiple representations of a document using templates that would be adaptable for different media sizes and capabilities.

For generating layouts, Jacobs et al. propose a workflow that couples different document creation tasks, such as:

- template adaptation;
- content selection;
- automatic picture cropping;

– document rendering;
– pagination; and
– low-level formatting (e.g. hyphenation, line-breaking, justification, kerning, etc.).

As in our work, they also use an optimization method for performing pagination, based on Plass's method [20], although admittedly a very complex adaptation that is tightly coupled with their rendering engine.

This coupled approach, however, leads to several shortcomings in our intended scenario. Their algorithm requires an intensive back-and-forth processing between the pagination algorithm, the rendering engine (for scoring the document formatting), and the constraint solver for the adaptive templates as well. Although the authors did not provide a more detailed performance analysis, we believe that the efficiency of such approach would not scale well for larger documents or mass production. The authors also point out that the adaptive template language is very hard to use when defining constraints between content placeholders.

Our approach, on the other hand, explores the separation between pagination and rendering, as this results in a more efficient workflow, and is also more adaptable to different workflows for custom documents. For instance, one may need different rules for content selection, or use different rendering engines for different media. Although our approach relies on static templates, the simpler template language makes authoring significantly easier.

More recently, Giannetti [9] described a model for linking elements from one or more streams of content over a set of page templates. The sequence of page templates used in the final document is determined by simple rules defined by the author, such as the repetition of a page template in even or odd pages, or only in the first page. Our approach uses the same streams of content, as for example there could be a stream of texts and pictures and another stream of advertisements to be placed across the publication, keeping balance between the amounts of each content. On the other hand, the choice of template is made optimally and not constrained by previously defined rules.

## 3 Problem statement and workflow

The goal of this work is to allow an author to produce a high-quality publication made of several pages based solely on its content and a set of templates for those pages. Therefore, the problem consists of splitting an input sequence of *contents* (e.g. headlines, texts, pictures, etc.) into a number of pages (possibly specified by the user). A set of *page templates* is provided, and for each page on the final document an appropriate template must be chosen to hold its content. A template is informally defined as a hand-made page design that holds one or more geometric placeholders, each representing a specific type of content, such as texts, pictures, advertisements, and others. Therefore, a pagination algorithm should include a *mapping* procedure that knows how to place content into a template.

Given a desired number $N$ of pages, it is necessary to split the content into $N$ parts such that for each part there exists a template able to hold it. Moreover, a sequence of content may fit in a template but produce wasted page space or the over/under filling of content inside a placeholder, causing an uneven distribution over a page. This accounts for the need for some *placement error* of a sequence of content into a template. Therefore, the splitting must be selected so that the placement errors are minimized (more details in Sect. 5.1).

Some natural requirements for a magazine-like publication are also assumed:

– The templates and their placeholders are rigid, as changes in their geometry are forbidden;
– The input order for textual content should be preserved;
– All pages must be filled entirely with content, including the last page;
– The selection of templates must be globally optimal according to some quality measure. For instance, all text placeholders must have a similar text density and the wasted space around pictures (due to differences in aspect ratio between pictures and placeholders) must be minimized.

As the content sequence will be split, text is modeled as a sequence of blocks (whole paragraphs, sentences, words, or even characters) that are discrete and indivisible objects. Therefore, text flows are handled after choosing an appropriate partitioning of content. This accounts for a *granularity* of content, where a finer division of text blocks will result in a better document, at the cost of run time performance. Such division of text objects is assumed to be performed prior to the pagination algorithm and represented in the input content accordingly. This will be discussed in more detail in the following sections, especially in Sect. 8.1.

The workflow is decomposed into independent steps as this solution suits a larger variety of workflows for producing documents. Separating concerns such as document pagination from rendering also results in a more efficient system, enabling the production of documents in a non-interactive setting. Basically, input is read from a sequence of content, decisions are taken about the layout and the generation of the final document is forwarded to a document processor such as InDesign. Figure 1 illustrates how input content is mapped to candidate templates and produces a sequence of pages with associated content.

Files describing the input content and the templates are fed into a *layout engine*, which breaks down content into

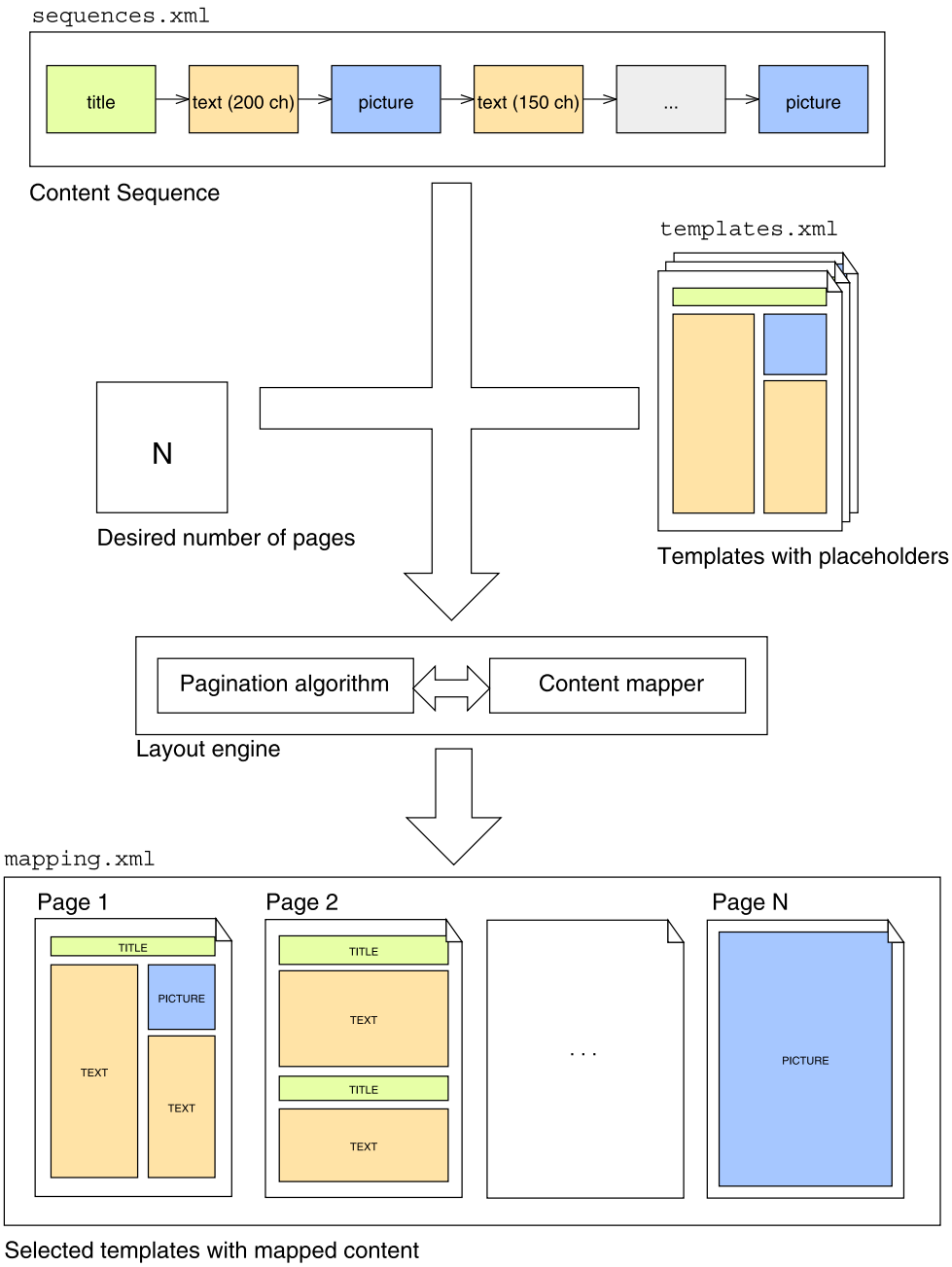



**Fig. 1** The workflow for the proposed pagination algorithm. An input sequence of content, a set of templates and the desired number of pages are fed to a pagination algorithm that produces an optimal mapping of content to page templates. The content is represented by the linked sequence of boxes in the upper part of the figure, where each piece of content has its type, denoted by the legend in each box. The corresponding XML files described in Sect. 4 are also indicated above each structure

pages and selects appropriate templates for each page as described in Sect. 5. Both input and output of the workflow are made through XML files described in the next section.

## 4 Document format

XML [18] files are used for the pagination and could be extended to a more complete document description language, such as PPML [1] or the higher-level Document Description Framework (DDF) [17]. Three XML formats are used for the different elements in our approach: content sequences, templates and the resulting mapping of content of a sequence into page templates. These will be briefly described in the following sections.

### 4.1 Content sequences

The *content sequences* file is responsible for representing the actual content to be presented on the document. Each file holds one or more content sequences, as multiple sequences are useful for representing elements that belong together such as news articles comprised of headlines, text and pictures. The `<obj>` notation was introduced to allow more

```
<sequences>
 <sequence>
  <obj type="picture">
   <picture> ...dsc02528....jpg </picture>
  </obj>
  <obj type="headline">
   <text> Donec odio neque, male... </text>
  </obj>
  <obj type="text">
   <text> Donec odio neque, malesu... </text>
  </obj>
  . . .
  <obj type="text">
   <text> Sed orci. Vestibulum a mi... </text>
  </obj>
 </sequence>
</sequences>
```

**Fig. 2** An example of a content sequence XML file

complex types of object to be defined, but in the examples from this paper only three types of atomic object are used: headlines, texts and pictures, discussed in Sect. 4.4. Figure 2 illustrates a content file.

### 4.2 Page templates

Templates use the format shown in Fig. 3 and their graphical representation is illustrated in Fig. 4. The template file contains several templates, each providing *placeholders* for different types of object. For simplicity, only rectangular placeholders (described by the `x`, `y`, `w`, `h` attributes) are allowed, but arbitrary polygons could also be supported. The `type` attribute defines what type of object that placeholder supports and text placeholders have a `capacity` that specifies how many characters of text they are able to hold. This number must be based on the selected base font size and spacing for rendering (see Sect. 6.2), and can be automatically calculated when authoring a template. Although the base font size is estimated, the rendering engine is later able to handle text more precisely, as discussed in Sect. 6. The text placeholder is the only one able to hold more than one object, as in the case of a text placeholder containing several paragraphs.

Adobe InDesign® was used to perform the rendering of documents, but it was also used for the authoring of templates using the format from Fig. 3.

### 4.3 Mapping

The mapping file (Fig. 5) is produced by the pagination algorithm and combines the content of the two previous files, defining relationships between placeholders from templates and content from the input sequences. Since each selected template is a page in the output document, they appear ordered in the mapping file as a `<page>` tag. The placeholders chosen for each page will have the associated content from

```
<templates>
 <template w="210" h="297">
  <obj type="headline"
       x="14.82" y="17.47" w="180.45" h="30.51"/>
  <obj type="headline"
       x="14.82" y="167.45" w="180.45" h="25.90"/>
  <obj type="text" capacity="2097"
       x="14.82" y="54.76" w="180.45" h="102.31"/>
  <obj type="text" capacity="1576"
       x="14.82" y="202.42" w="180.45" h="77.61"/>
 </template>
 <template w="210" h="297">
  <obj type="picture"
       x="12.70" y="12.70" w="184.60" h="271.60"/>
 </template>
 <template w="210" h="297">
  <obj type="headline"
       x="16.93"  y="19.58" w="175.15" h="20.11"/>
  <obj type="text"
       x="16.93"  y="49.41" w="92.60" h="227.34"
       capacity="2367"/>
  <obj type="text" capacity="1201"
       x="120.88" y="125.41" w="71.21" h="151.34"/>
  <obj type="picture"
       x="120.88" y="49.41" w="71.21" h="69.00"/>
 </template>
</templates>
```

**Fig. 3** Sample templates XML file

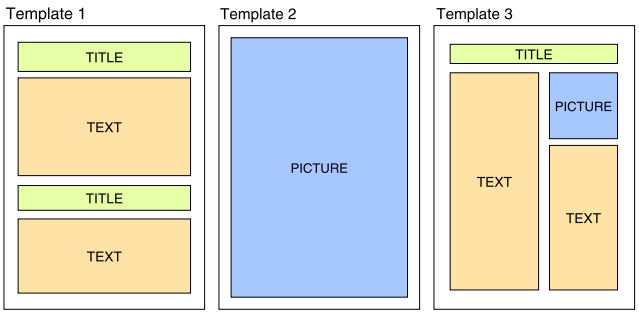



**Fig. 4** The templates described in the XML file on Fig. 3

the content sequences file as child elements as well as their geometric information. Text placeholders may also contain more than one text object.

### 4.4 Introducing new objects

As templates handle the same types of object as contained in the input sequence and a placement error can be measured for each type (see Sect. 5.1), it is possible to extend the model and introduce new types of content as necessary. For instance, publications may define an *advertisement* type and some of the available templates will be able to hold such objects. As another possibility, using the *picture* type could be avoided altogether by introducing two new types *smallpic* and *bigpic* to be used only in specific places on the tem-

```
<magazine>
 <page w="210" h="297">
  <obj type="picture" x="12.7" y="12.7" w="185" h="272">
   <picture> ...dsc02528....jpg </picture>
  </obj>
 </page>
 <page w="210" h="297">
  <obj type="headline" x="16.9" y="19.6" w="175"  h="20.1">
   <text> Donec odio ne... </text>
  </obj>
  . . .
 </page>
 . . .
 <page w="210" h="297">
  . . .
  <obj type="text" x="14.8" y="202" w="180" h="77.6">
   <text> Donec vulputate... </text>
   <text> Donec odio nequ... </text>
   <text> Sed orci. Vest...  </text>
  </obj>
 </page>
</magazine>
```

**Fig. 5** Output XML file containing the resulting pagination and content mapping

```
<!-- A captioned picture in a content sequence file -->
<obj type="captioned-picture">
 <picture> Picture source </picture>
 <text>    Caption text   </text>
</obj>

<!-- A captioned picture in a template file -->
<obj type="captioned-picture">
 <obj type="picture" x="12.7" y="12.7" w="185" h="272"/>
 <obj type="text" x="12.7" y="290" w="185" h="10" capacity="300"/>
</obj>
```

**Fig. 6** Defining a captioned picture object

plates, gaining more control of the final look of the publication.

Consider that a new type is defined, representing a captioned picture. The XML syntax used in the template and content sequence models would be as illustrated in Fig. 6. Note that a composite type may have many different geometric representations, depending on how each instance is defined in the templates file.

Composite elements can also be used to prevent related objects to be split apart in two different pages by the pagination algorithm. For instance, an `article` object type may be composed by a headline, two columns of text and a picture for that article. The entire article will appear as a single object to the pagination algorithm, and as a consequence its contents will appear together in the final document.

## 5 The pagination and mapping algorithms

For each content sequence the basic algorithm finds the best way to split it into pages in such a way that each selected template produces a minimal mapping error. The optimization function selects the best set of templates based on the method adapted by Oliveira in [19] from Skiena [24], which can be defined as

$$P_{n,p} = \begin{cases} \sigma(1,1) & n = 1 \\ \sigma(1,n) & p = 1 \\ \min_{1 \le i \le n-1} \max_T (P_{i,p-1}, \sigma(i+1,n)). \end{cases} \tag{1}$$

This recurrence can be implemented as a dynamic programming algorithm that fills a table $P_{n,p}$ holding information to optimally divide a sequence of $n$ elements into $p$ page templates.

In (1), $\sigma(i, j)$ is an *error* function that returns the minimal error obtained by the templates from the set $T$ of templates when holding sub-sequence $[i, \ldots, j]$ of contents. The $\sigma$ function must also keep information about which template is the best choice for the given interval, for further use. The error function is given in Sect. 5.1.

Each case of the recurrence in (1) may be described as follows:

- When $n = 1$, only the first element of the sequence remains, so it has to be placed in a single page template, which is selected by the $\sigma$ function with $\sigma(1, 1)$;
- When $p = 1$, all elements of the sequence $1, \ldots, n$ must be placed in a single page template, selected by $\sigma(1, n)$;
- Otherwise, an optimal page break must be found, so the recurrence attempts to split the content sequence into two groups at every possible break point in the sequence $1, \ldots, n-1$, and solving the problem recursively at the left side of each break point. By minimizing the largest error $\sigma$ between the left and right sides of the sequence, it is possible to find a sequence of page breaks that minimizes the global error from $\sigma$ for every page $1, \ldots, p$.

During template evaluation, it may happen that some element in the sequence does not fit in the candidate template, or some placeholder from that template may be left empty. In that case, this template must be rejected, by attributing to it an infinite error ($\sigma(i, j) = \infty$). The pagination algorithm will then select different sequences of elements for mapping. If no valid sequences can be found, the algorithm will fail to provide a solution. Therefore, a different or larger set of templates must be provided to handle the same input sequence. If input order is not important, reordering the sequence can also be attempted to yield a valid solution.

Figure 7 illustrates a scenario where templates are being selected for a set of attempted page breaks, during the evaluation of the recurrence in (1).

Figure 8 shows a sequence of pages produced from the templates in Fig. 4 with mapped content and ready to be sent to a rendering stage.

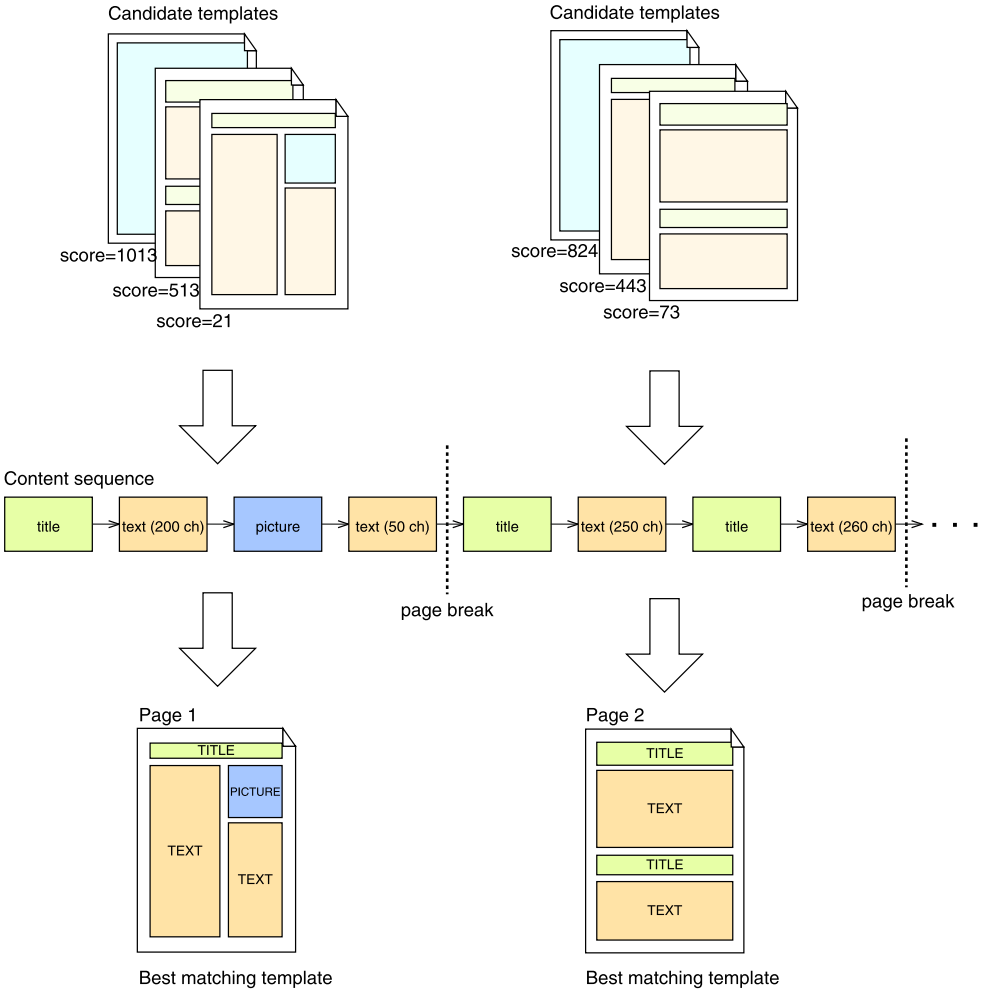



**Fig. 7** Instance of the pagination algorithm, evaluating the set of templates for each attempted sub-sequence of content

### 5.1 Template scoring and mapping

To select the best template for a sub-sequence of contents $[i, \ldots, j]$, some *error measure* that evaluates how well the content fits one template must be provided. From that it is possible to pick the best matching template among the candidates. The pseudo-code for such an algorithm is illustrated in Fig. 9. It tries every template $t$ in $T$ and by attempting to map the content sequence $[i, \ldots, j]$ to it (i.e. calling the `allocate(t,i,j)` function), it is able to measure the placement error and minimize it. During evaluation, information about the best template found for each pair $(i, j)$ is kept as this enables us to recall the selected templates after an optimal pagination has been found by the main algorithm.

The actual measurement of the placement error is performed by `allocate(t,i,j)`, which receives a template $t$ from $T$ and a sub-sequence $[i, \ldots, j]$. This procedure is described below.

### 5.2 Allocating elements to placeholders

Once an interval $[i, \ldots, j]$ of elements has been selected for placement, it is necessary to map this interval to the candidate template and measure the placement error. Templates that cannot match the sequence are discarded before attempting any mapping of content.

For text mapping, the placement error for one or more texts totaling $N_t$ characters in a text placeholder $c$ with total text capacity $N_c$ and text area $A_c$ is defined as

$$A_c * \left( \max\left( \frac{N_c}{N_t}, \frac{N_t}{N_c} \right) - 1 \right), \tag{2}$$

which penalizes over/under filling the placeholder.

To map pictures, we try every ordering of pictures on the page to minimize the wasted space by aspect ratio differences between the pictures and their placeholders, since no automatic cropping or non-proportional scaling is performed. However, a picture is only moved out of order if it

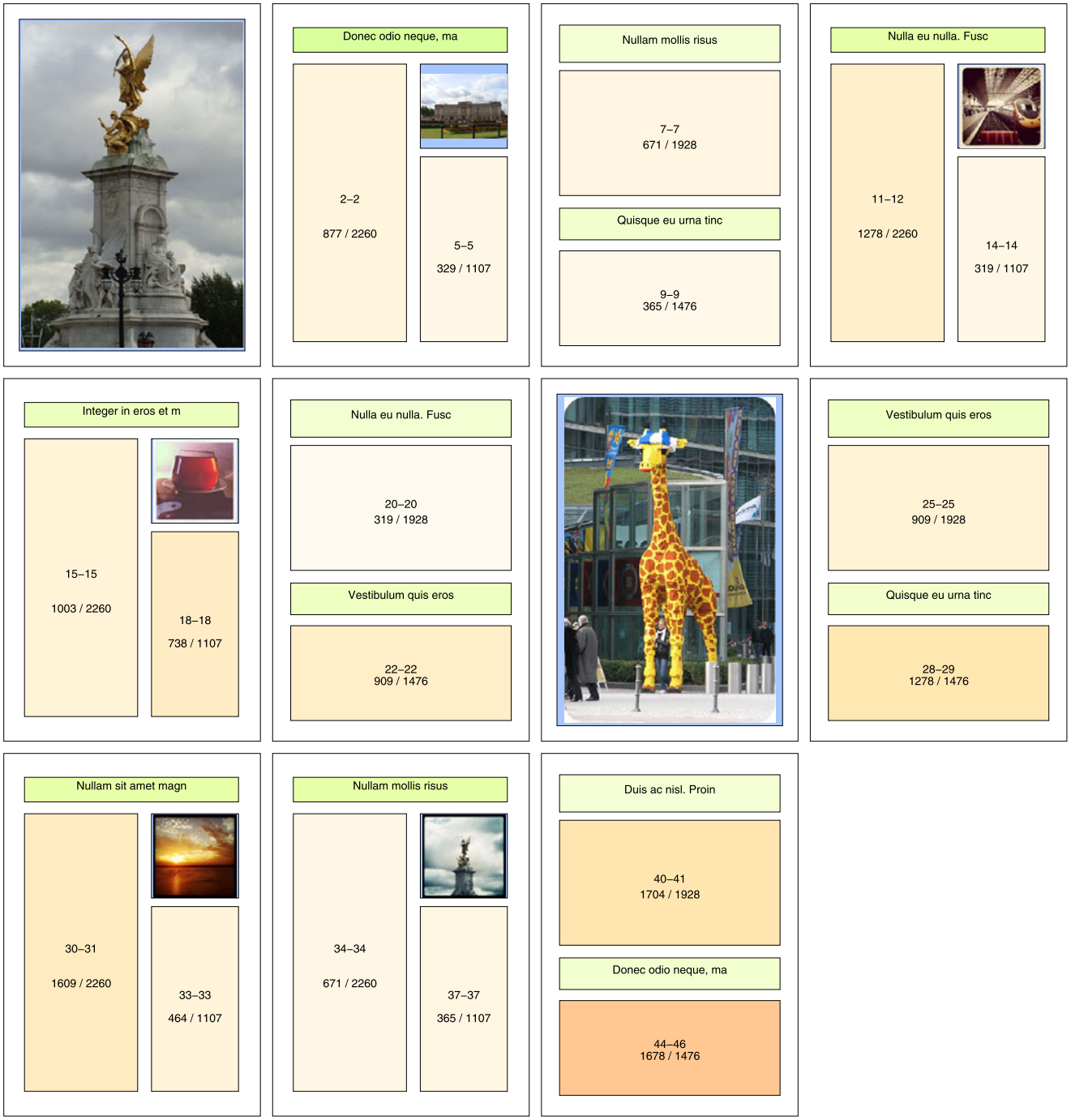



**Fig. 8** Example of a result from the pagination algorithm, matching a sequence of templates to the input content. The numbers on the text placeholders indicate which texts in the sequence have been allocated to that placeholder, as well as the ratio of the amount of text by the placeholder's capacity (i.e. how "filled" a placeholder is)

is unrelated to the other contents on the input sequence. The coupling between pictures and other pieces of content can be specified in the input content by the use of composite types, described in Sect. 4.4.

The placement error for a rectangular picture $p$ having aspect ratio $\Delta_p$ in a rectangular placeholder $c$ with aspect ratio $\Delta_c$ and area $A_c$ is

$$A_c * \left( \max\left( \frac{\Delta_c}{\Delta_p}, \frac{\Delta_p}{\Delta_c} \right) - 1 \right), \tag{3}$$

which penalizes aspect ratio differences between a picture and its placeholder, proportionally to the area of the placeholder.

```
1. function σ(i, j)
2. {table is a global data structure filled with the best
   matching templates for each pair (i, j)}
3. if table[i, j].memoized then
4.    return table[i, j].error
5. end if
6. minerror ← ∞
7. bestt ← nil
8. for each template t in T do
9.    error ← allocate(t, i, j)
10.   if error < minerror then
11.      minerror ← error
12.      bestt ← t
13.   end if
14. end for
15. table[i, j].memoized ← true
16. table[i, j].template ← bestt
17. table[i, j].error ← minerror
18. return minerror
```

**Fig. 9** Template scoring and selection function

To calculate the final error for the template, the maximum placement error is used among all placeholders of the template. The pagination algorithm then uses this error to find the solution that is the global minimum, effectively minimizing the maximum placement error.

## 6 Rendering with InDesign

After selecting appropriate page templates and mapping content into them, it is necessary to render the actual content in the templates and perform the lower level text formatting to generate the final document. Figure 10 depicts how the rendering step is integrated to the workflow from Fig. 1.

For rendering, the pagination algorithm was integrated to Adobe InDesign®, a standard application in document processing and publishing systems that handles lower-level formatting. The integration was performed with functions inserted through InDesign's scripting system [2] using the JavaScript language [7]. These functions perform the following steps:

1. The mapping XML file (Sect. 4) generated by the pagination algorithm is read by InDesign;
2. Each page is assembled with its placeholders and their corresponding content from the content sequences;
3. Each type of element (i.e. text, headlines, pictures, etc.) is rendered according specific rules, as described in the following sections.

Additionally, for composite elements (a captioned picture for instance, see Sect. 4.4), the rendering is performed recursively inside the element's corresponding placeholder on the page. The following sections describe how each type of element as described in the document model is rendered.

### 6.1 Rendering headlines

Headlines are basically single lines of text that must fit inside their associated placeholder in the page template. It is assumed that the placeholder size is the intended size for its content as well, therefore typographic adjustments are made by changing the font size, so that the headline fits its placeholder entirely.

### 6.2 Rendering texts

To place text objects in InDesign, a two-step procedure was used: flowing and fitting. Fitting adjusts font sizes to fit a block of text to the placeholder using the same procedure applied for headlines. Flowing means making a single block of text span one or more placeholders, for example when the text of one article spans two columns on a page. InDesign allows a block of text to flow across logically linked text placeholders, and thus it is desirable to take advantage of such feature, for more realistic layouts. But since the pagination algorithm cannot make text objects flow between placeholders, a simple heuristic was used as described by the steps below:

1. The text placeholders are sequentially linked in each page template and across the pages. When the document contains a single sequence of content, a simple heuristic is to consider each headline occurring in the sequence as the beginning of a new "article". Thus, every text placeholder is connected sequentially (as given by the input order) until a new headline appears. Then a new flow begins and the process is repeated. If the content is already split into several sequences, another approach is to simply associate every text placeholder to each sequence.
2. After the flows are created, the font size for each flow is adjusted across text placeholders belonging to this flow, therefore completely filling up every text placeholder. Since this adjustment is performed for a whole series of placeholders, the font size is changed only very slightly, and is uniform across the whole flow.

It is also possible to avoid text flows at all, by adjusting each text placeholder separately. However, if text densities change too much between adjacent placeholders, the font sizes will be visibly different, creating an unpleasant effect.

Flowing information could also be explicitly defined in the template language, but such idea is not explored in this paper. However, the template language may easily be extended to support explicit text flows.

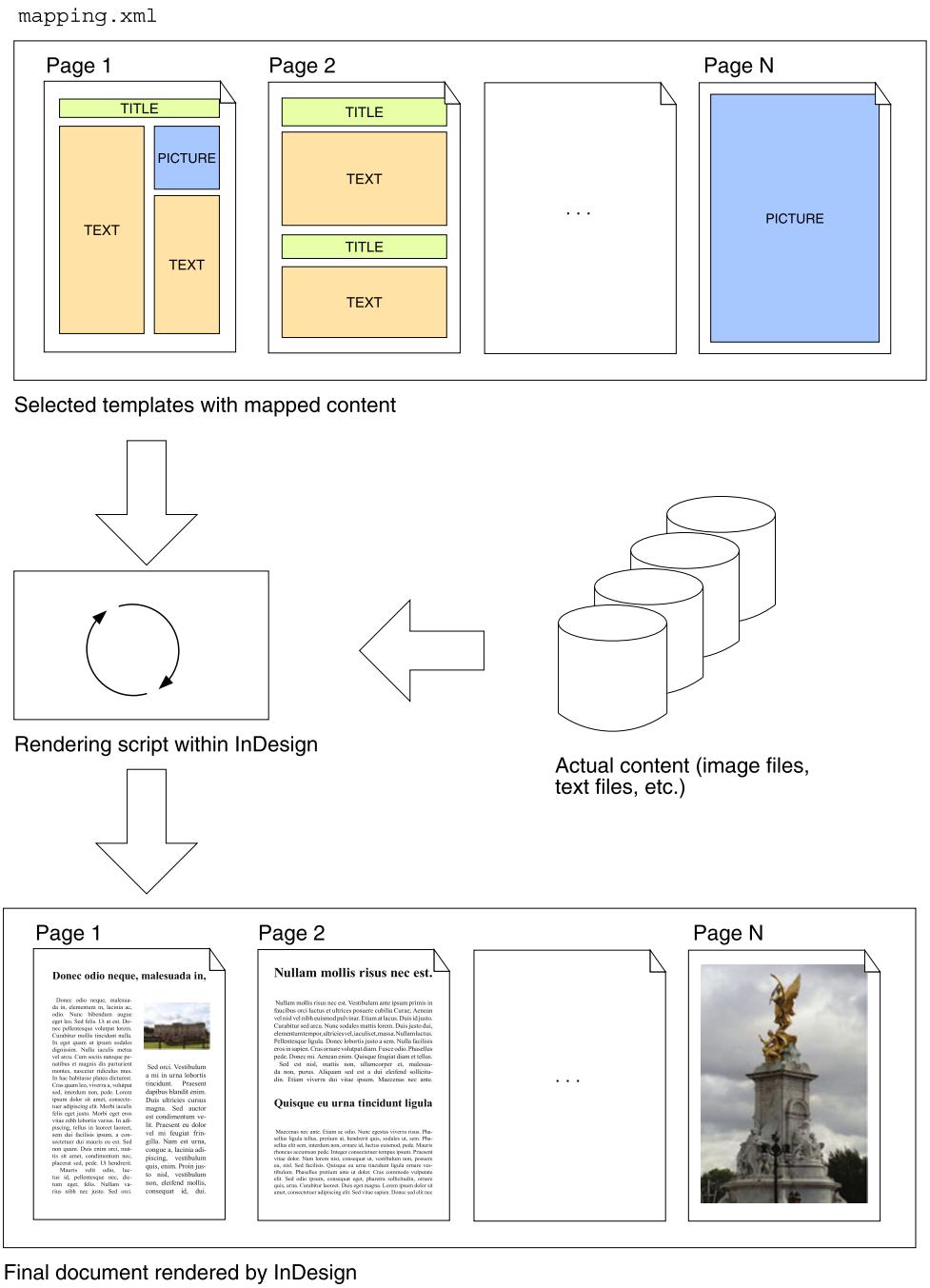



**Fig. 10** The workflow for performing the actual document formatting step, using InDesign

To illustrate text flows in InDesign, Fig. 11 shows an extreme example with two versions of a one-page document. This document is comprised by two text articles containing two text objects each, where their lengths range from very small to very large, creating an unpleasant effect. Figure 11(a) illustrates the document where no flow strategies have been applied, whereas Fig. 11(b) shows that texts inside each article have been linked, resulting in a more homogeneous font size for each article.

### 6.3 Rendering pictures

To place a picture in a placeholder, one of the picture's dimensions (width or height) is scaled to that of the placeholder, and the other dimension is set so that the picture's aspect ratio is preserved and no part of it is clipped. Automatic cropping of the picture is not attempted, although it would be possible to integrate such a feature.


(a) No linking between text placeholders.

(b) Linking text placeholders at each headline occurrence.

**Fig. 11** Two versions of a one-page document, showing the use of text flows in InDesign

**Table 1** More detailed description on the content datasets

| Dataset | Text objects | Pictures | Avg. text length |
|---|---|---|---|
| `folha` | 2420 | 318 | 660 chars. |
| `lipsum` | 14 | 84 | 1212 chars. |

## 7 Results and discussion

This section presents results produced by the algorithm using empirical data and InDesign to generate ready-to-print documents.

In Sect. 7.1, the method is compared to a simpler approach, where quantitative measures show that the pagination algorithm scores significantly better. Consequently, possibly longer running times are justifiable if better-quality documents are required. Section 7.2 discusses the performance of the pagination algorithm, comparing it to a simpler but efficient method.

A qualitative evaluation is presented in Sect. 7.3, where output documents are compared to others generated with a simpler pagination approach, as well as a comparison between the proposed method and a real-world magazine, although some limitations must be considered.

### 7.1 Minimization of worst error and density variation

To evaluate the solution and its benefits, test instances were generated and the results using the proposed algorithm were compared against a simple first-fit method. This was implemented as a greedy algorithm which attempts only a single choice for content placement. It works as follows:

1. The input is the same as before: a content sequence and a set of templates;
2. At each step, every template is tested against the current sequence of content, in order to evaluate how much content can be consumed by this template;

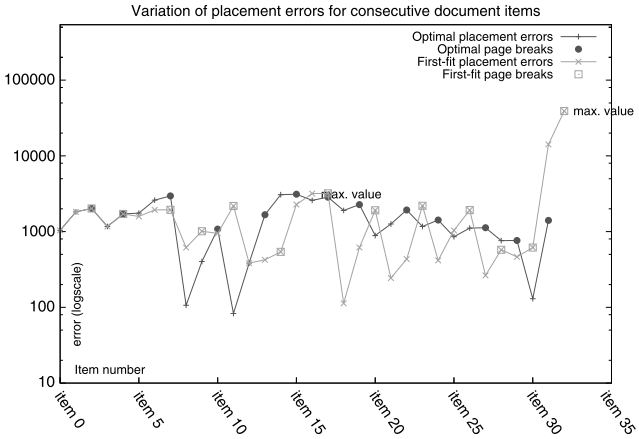



**Fig. 12** Comparison of placement errors between the proposed approach and a first-fit pagination

3. The template that uses the most content with the least amount of error is selected and the sequence of content is reduced.
4. The algorithm repeats from step 2 until the content sequence is empty.

Given the greedy nature of a first-fit pagination strategy, sometimes there will be no way to fill up a template, leaving empty placeholders in the output document. For the purposes of this evaluation, this is going to be allowed for the first-fit method (when it fails to find an adequate solution) by assigning very large errors (more precisely, $10^6$) to unused placeholders.

As the optimal pagination algorithm has a wider range of available solutions to search than simpler pagination strategies, it is able to select a solution that is more homogeneous (according to the objective function from (1)), where the content is well distributed over the document pages.

#### *7.1.1 Generating test instances*

For generating the test instances, two datasets containing an assortment of texts and pictures with varying aspect ratios were used:

1. The `folha` dataset (obtained from a Brazilian newspaper's RSS feed),[4] which is comprised by a large number of text articles (mostly short) and pictures;
2. The `lipsum` dataset, which is smaller and contains larger, randomly generated text articles.

Table 1 shows more details about the datasets.

#### *7.1.2 Single-instance results*

Figure 12 illustrates the placement error for a test case using both pagination methods, according to the error functions

[4] http://www1.folha.uol.com.br/folha/informatica/ult124u16829.shtml.

from (2) and (3). The results indicate that document items are more evenly distributed across the pages by our method, resulting in a more aesthetically pleasing document [12]. The test instance is a document containing approximately 100 text objects. The template set used for this test is shown in Fig. 13 and the resulting mapping and pagination for both methods is shown in Fig. 14. A text placeholder in the templates can be filled up with any number of text objects, and the graph in Fig. 12 shows the placement errors for each placeholder (i.e. the $x$-axis).

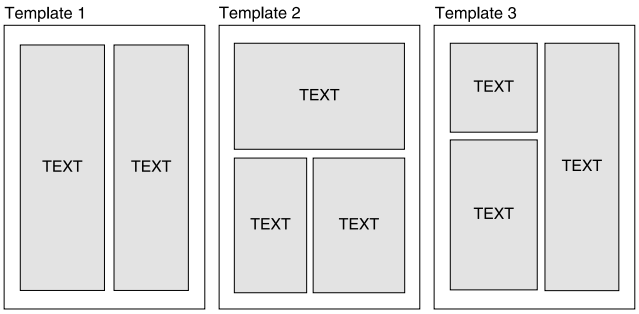



**Fig. 13** Template set used for the test presented in Fig. 12

Another important detail in Fig. 12 is that the placement error from the first-fit pagination increases abruptly at the end. This is a well-known tendency for the first-fit method, as it uses the content sequence to fill up pages until there is a small sequence left at the end, causing the last page to be under-filled. This does not happen in the method proposed in this paper, as it guarantees an even distribution of content.

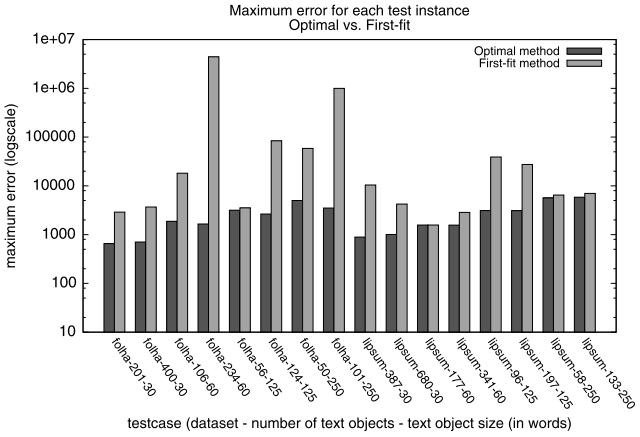
### *7.1.3 Multiple-instance results*

More results are presented for several test instances, using both datasets from Table 1. However, only text objects were used, because the inclusion of headlines and/or pictures would cause the first-fit pagination to fail more often and result in worse choices. Test instances were generated with the number of text articles ranging from 50 to 100 broken in smaller objects of sizes from 30 to 250 words each.

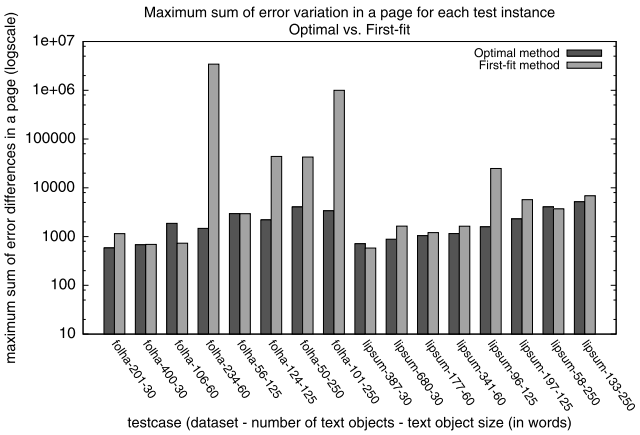
(a) First-fit mapping and pagination.

(b) Optimal mapping and pagination.

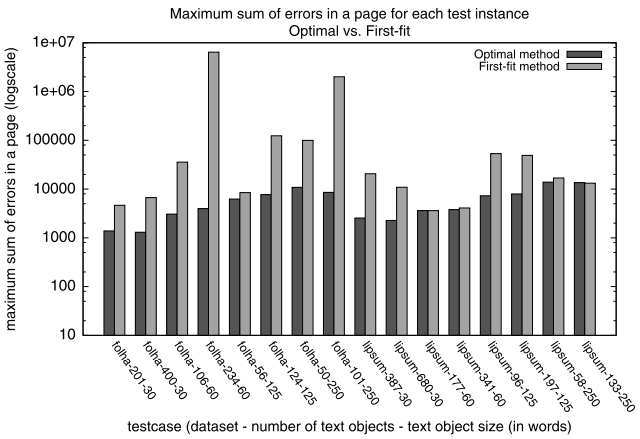
**Fig. 14** Mapping and pagination of both methods for the test presented in Fig. 12. The *gray shades* indicate how much a placeholder is filled with text




(a) Maximum error.




(b) Maximum error variation in one page.




(c) Maximum sum of errors in one page.

**Fig. 15** Results of both methods for more test instances using three different quantitative measures

Figure 15 shows the results for both methods using three different measures: the worst error over all placeholders in the document (Fig. 15(a)); the maximum error "variation" (i.e. sum of error differences) in a single page (Fig. 15(b)); and the maximum sum of errors in a single page (Fig. 15(c)).

### 7.2 Performance

The system described in Sect. 3 was implemented using Java. Given real-world time constraints, the pagination algorithm (see Sect. 5) had to be implemented using dynamic programming, for producing output in acceptable running times. The allocation algorithm from Fig. 9 also uses memoization in order to store the best template selections for each different sequence of elements.

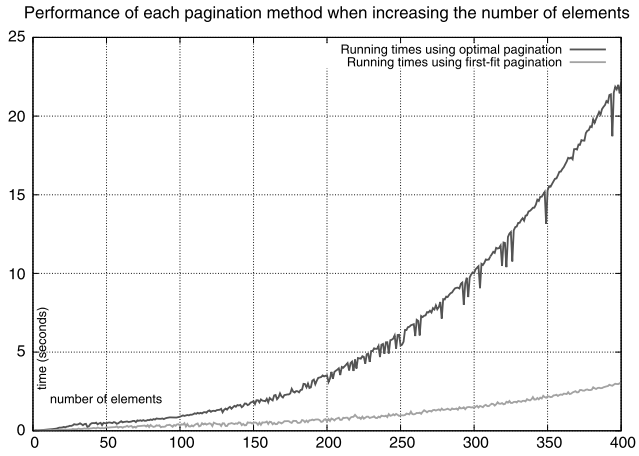
Using these optimizations, the worst-case performance of the pagination algorithm is bounded by $O(n^3)$ when the number of pages is unknown and $O(k\,n^2)$, when a number $k$ of pages is defined, where $n$ is the size of the input sequence. The cost for memoizing template choices depends on the number of templates and the methods for allocating elements to placeholders, described in Sect. 5.2. In the worst case its asymptotic bound is $O(|T|\,n!)$, where $|T|$ is the number of available templates. This is due to the allocation method for pictures, which requires a search for every possible ordering to find the best match in pictures' aspect ratios. If reordering is not allowed, the bound drops to $O(|T|\,n^2)$. In practice, however, this cost is low because templates are checked for matching before attempting allocation and usually they contain just a few placeholders.

Two performance tests were made to compare the first-fit and optimal algorithms, shown in Fig. 16. Figure 16(a) shows how the running times behave when using an increasing number of textual elements but keeping the number of templates constant (50 in this case). Figure 16(b) on the other hand shows a different behavior when the number of templates is increased and the number of elements is kept constant (100 textual elements).

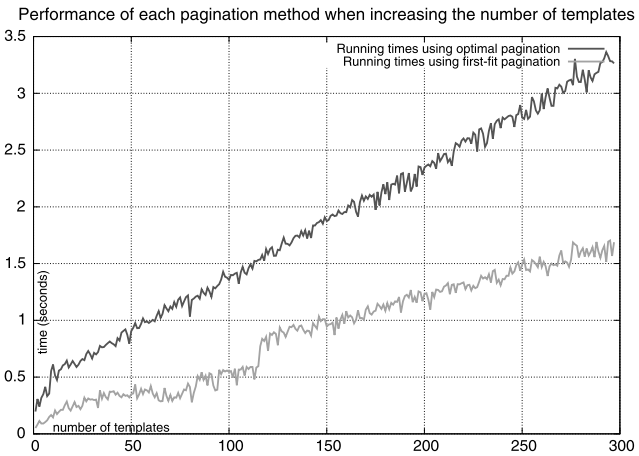
Although the performance degradation in the optimal algorithm can be severe when processing very large sequences of content, we found that the algorithm performs well on typical input sizes. For instance, a magazine comprised of 250 elements (20 % pictures and 80 % text) matched against 500 templates was generated in under a minute, running on an Intel® Core™ 2 Duo 1.86 GHz machine with 2 GB of memory. On the other hand, increasing the number of available templates does not seem to impair performance quickly, according to Fig. 16(b).

### 7.3 Camera-ready results

Though no user studies were performed in this paper for qualitative assessment, camera-ready PDF files produced by the workflow from this paper are presented below. The documents are a proof-of-concept for the pagination and mapping algorithms, and no further attempts are made to enhance the templates with features such as visual styles, fixed elements on a page, page imposition, and others. Therefore, pages contain only the variable content that has been




(a) Curves when increasing the number of elements.




(b) Curves when increasing the number of templates.

**Fig. 16** Performance comparison between the optimal and first-fit pagination algorithms when increasing number of elements or templates

mapped to them. However, a visual comparison with a real-world example is still useful as it provides basis for future enhancements.

Templates used in this test were not created by graphic designers, but were generated automatically using different configurations of a set of boxes of fixed geometry. Approximately 300 different templates were generated using this method.

It is important to mention that the templates were not generated with aesthetic concerns in mind (i.e. no attempts were made to produce aligned or well distributed containers), but rather to demonstrate how the pagination algorithm produces balanced distribution of content among pages.

The test document was crafted to mimic a magazine publication from our university. For the comparisons, only pages that were simple enough to contain mostly headlines, texts and pictures have been selected. Texts and pictures from such pages were extracted, but details such as headers and footers were left out, given that the templates used here are quite simple. Figure 17 presents a document produced by the workflow. The documents to be compared to this one will be shown in the two following sections.

#### 7.3.1 *A real-world magazine*

Figure 18 shows 15 sample pages of a magazine from our university.[5] Although the quality of the templates used in this paper is simpler when compared to the pages from the magazine in Fig. 18, an important detail to notice is that in Fig. 17 the pagination algorithm was able to preserve the same content structure from the real magazine (i.e. the sequence and grouping between titles, texts and pictures), without markedly increasing the number of pages. Another important detail is that all pages are completely filled, including the last one.

#### 7.3.2 *First-fit comparison*

The results from the optimal pagination method were compared to the first-fit heuristic described in Sect. 7.1. Figure 19 shows the same test document from Fig. 17, but generated from the first-fit method.

The first-fit algorithm generated 40 pages, and only the first 20 are shown in Fig. 19. This was almost twice the number than generated by the optimal method, due to the regions left empty by the first-fit method. As it can be seen, this can impair both document readability and aesthetics.

As discussed in Sect. 7.1, due to the poor set of choices from the first-fit method, some placeholders could not be filled and little content has remained in the last page, as this method is unable to attempt a more balanced division of content to pages.

### 7.4 Discussion

Results indicate that the proposed algorithm is able to generate high-quality solutions in practice. Given that the evaluation was made by comparing the algorithm to a first-fit heuristic, one may point out that an exhaustive method will always win against a simple heuristic. We chose a simple heuristic due to a lack of options, as we are not aware of any other openly available method for performing automatic pagination and template selection for variable content. Aesthetic evaluation [12] would also be of limited value, given that templates are rigid, so the aesthetic measures would be more dependent on the templates' design than on the content mapping itself.

[5] Obtained from http://www.pucrs.br/revista/pdf/0149.pdf.

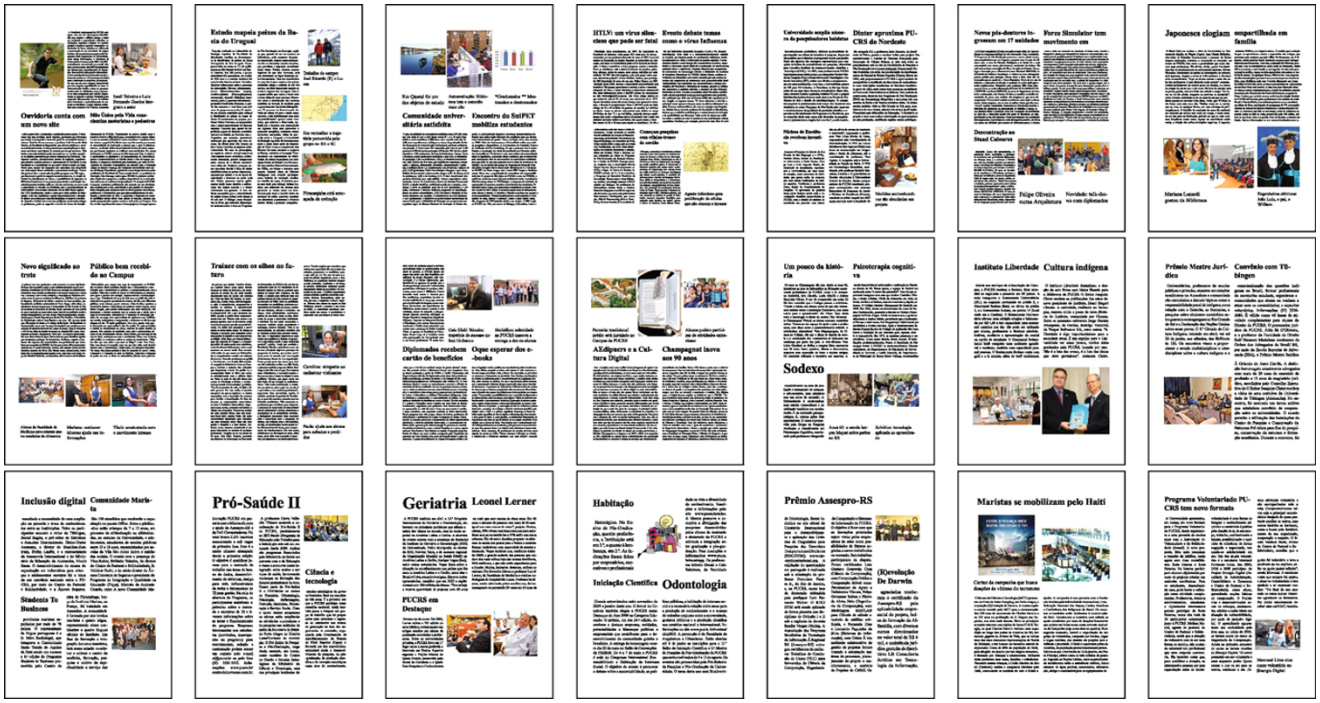


**Fig. 17** Test document as produced using our workflow with optimal pagination. It mimics the document from Fig. 18

**Fig. 18** Sample pages of a real-world magazine publication

## 8 Tuning documents

The pagination algorithm offers some extra advantages to solve the pagination problem in a more general way:

- Fine tuning of text granularity (i.e. breaking input text into smaller elements, such as paragraphs, sentences, words, etc.);
- Automatic selection of the optimal number of pages for the publication.

These extensions are discussed in the next sections.

### 8.1 Changing text granularity

One apparent problem with the solution is that, contrary to other pagination methods that consume text from input as a continuous stream, texts are handled as discrete blocks to be mapped into a single template placeholder. As discussed in Sect. 3, this causes two problems:

- It is not possible to make text objects flow across different pages;
- Large text objects may be forced to fit into a small placeholder, resulting in a bad distribution of text densities over the document, damaging the balanced distribution requirement for the proposed algorithm.

Both problems can be circumvented by breaking large text objects into smaller units prior to sending them to the

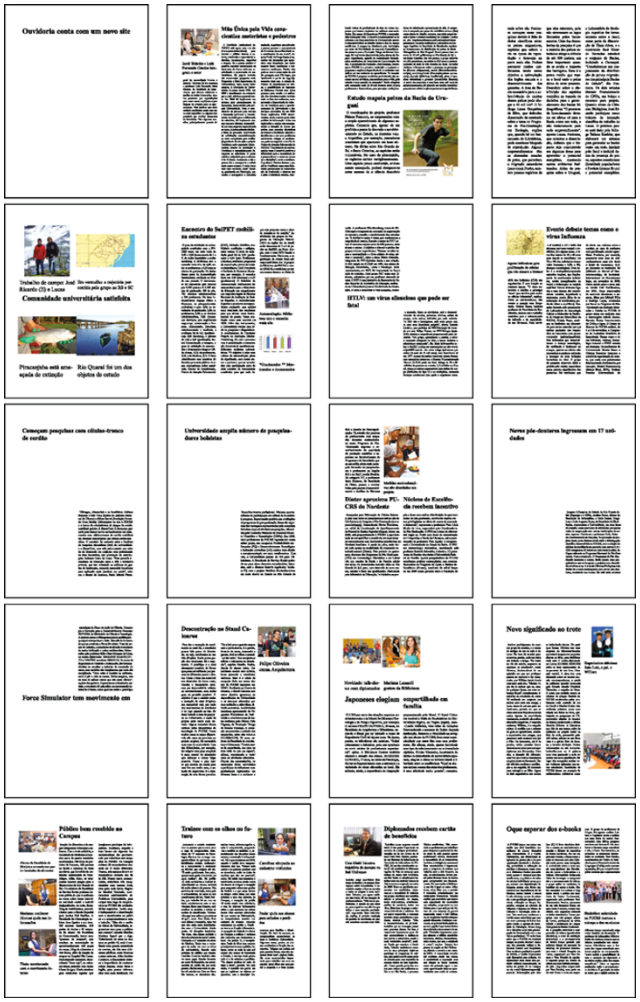


**Fig. 19** The same document from Fig. 17, but generated using a first-fit pagination method. Only 20 out of the total 40 pages are presented here

pagination algorithm. This is a feature that can be made transparent to the user, given that the smaller text objects still preserve their input order, and will appear more evenly spread across the pages.

To test this strategy, two test cases are provided: the first example used 50 different text objects, and the second one had each of the 50 objects automatically split as a new text object at every 76 words, resulting in 150 smaller objects. Figure 20 illustrates the results obtained with these two approaches. In Fig. 20(a), the gray shades in each placeholder represent the density information for that placeholder (text area/placeholder capacity). Placeholders with higher ratios (and darker shades) will hold too much text, resulting in over-filled placeholders along with under-filled ones. Figure 20(b) has placeholders with similar shades, showing that text is more evenly distributed on the page and resulting in a better document.

One drawback of this strategy is that having too many small text objects may result in a longer execution time of the algorithm, as discussed in Sect. 7.2. However, tests show that it is not necessary to increase granularity too much to obtain a good document, as can be seen from Fig. 20. In this case, a set of 20 templates were used, and the difference in running times was less than a second, as shown in Table 2.

### 8.2 The optimal number of pages

Given the recursive nature of the splitting algorithm as it constructs an optimal pagination of $p$ pages by solving an optimal pagination of $p-1$ pages first, it is possible to select the optimal number of pages by solving the problem starting from the longest possible magazine, that is, having one element on each page, and the solution of this problem will include all shorter versions of it (including the extreme situation of all elements on a single page), so that it is only necessary to choose the number of pages that produces the minimal error.

Figures 21 and 22 illustrate PDF output from the same test case. Figure 21 shows a document with 12 required pages. Figure 22 on the other hand has 8 pages and was optimally selected from the solutions already computed by the pagination algorithm.

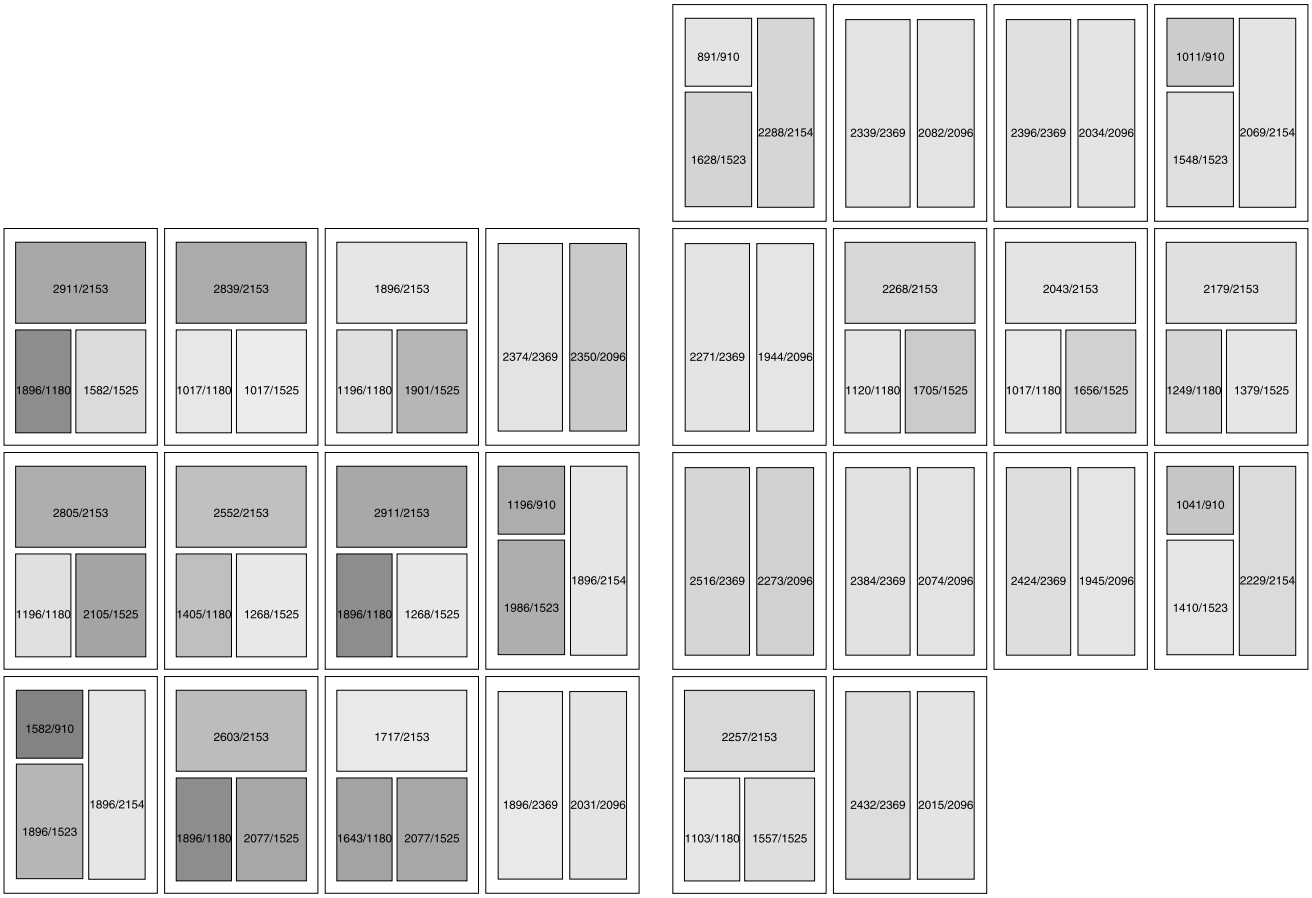
## 9 Conclusions

This work presented a new algorithm for the construction of personalized documents using page templates and optimally choosing the best templates to hold the content. In addition, the algorithm can be used to find the best number of pages and the best templates for a given amount of content. Thereafter, a description of the document is sent to a standard tool from the printing industry and graphical quality comparable to high-end publications was achieved. This approach can be easily integrated into workflows that require automatic pagination and mapping of content, such as VDP or self-publishing services.

Although the quality of the solutions cannot be compared to hand-crafted publications, a middle-ground is provided between purely automated solutions and high-quality, non-personalized documents.

The self-publishing scenario described earlier could also benefit from an automated approach. For instance, authors with no experience in graphic design could simply submit their content and select a "style" for his/her magazine (i.e. different sets of templates), leaving the actual layout production to the publishing service.

We believe that an automated process for the creation of high-quality personalized publications will leverage the publishing businesses, marketing-oriented VDP and publishing of web-driven content. We can envisage several applications for this work:




(a) Pagination with large pieces of text, resulting in too much difference of text densities.

(b) Breaking text into smaller pieces yields a better distribution, at the cost of a longer execution time and more pages.

**Fig. 20** Tuning text granularity

**Fig. 21** Example of a magazine generated with the proposed method, composed by 12 pages as requested by a user

**Table 2** Running times for low and high granularities of text objects

| Text splitting | Objects | Time (sec) |
|---|---|---|
| Low granularity | 50 | 0.206 |
| High granularity | 150 | 0.898 |

– Create high-quality publications out of web-driven dynamic content, such as personal blogs and RSS feeds [6];
– Automate self-publishing services, such as MagCloud® from HP;
– Enable pagination on VDP workflows for personalized catalogs [23].

The contribution presented in this work is the core of a larger workflow, of which we presented only a simplified model, leaving room for several possible improvements. For example, it is not possible to toggle on or off optional element placeholders or convert between elements (placing a picture on a text placeholder if necessary, for example). Moreover, the template selection method searches for mul-

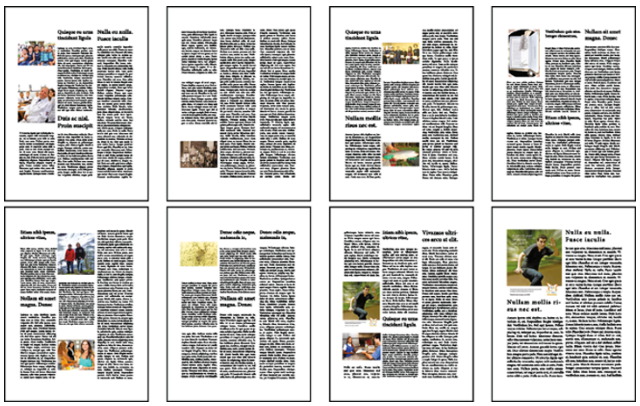


**Fig. 22** The same magazine shown in Fig. 21, but with its number of pages optimally selected and computed by the pagination algorithm

tiple combinations of picture placement on a page, which can be costly for the generation of picture-driven documents such as photo albums, and could be disallowed for a faster algorithm. Images could also have scaling constraints added to the scoring function from (3) (Sect. 5.2). For example, busy pictures would only be considered for placement in larger containers. This would result in a more efficient method with better results as well.

To handle these issues, the use of adaptive grid-based templates, as suggested by Lin [16] and Jacobs et al. [14] could be an alternative. However, the difficulty in automatically specifying relations between elements in adaptive templates would hinder flexibility in automated workflows. We believe that in the near future the construction of personalized publications will be more content-driven, so devising a set of complex rules for broad scope content selection would be a challenge.

Regarding the pagination algorithm, while we do not consider line-breaking and justification issues [14], the decoupling between rendering and layout evaluation is cleaner and encourages re-usability of the method. What is needed today is not a complete monolithic system, but rather the easy integration with other personalized workflows, to reach a larger scope on web-driven publishing. RSS delivery [6] and online photo albums [3] are a good example of possible integrations.

Finally, one interesting improvement that could be made to the pagination algorithm is the handling of solutions with too many repeated instances of the same templates, which usually result in a monotonous publication. A simple mechanism to count the number of times a template has been used and penalize it in the error function would be sufficient to allow for more variability on the appearance of the document, but further investigation is still necessary.

**Acknowledgements** This paper was achieved in cooperation with Hewlett-Packard Brasil Ltda. using incentives of Brazilian Informatics Law (Law n°. 8.2.48 of 1991).

## References

1. Podi, *Personalized print markup language (PPML) 2.2* (2008). http://www.podi.org/
2. Desktop publishing software—adobe indesign cs4 (2010). http://www.adobe.com/products/indesign
3. Photo books—create a wide variety of photo books using your favorite photos from snapfish (2010). http://www2.snapfish.com/photobookcategory/COBRAND_NAME=snapfish
4. Bagley SR, Brailsford DF, Ollis JA (2007) Extracting reusable document components for variable data printing. In: DocEng'07: Proceedings of the 2007 ACM symposium on document engineering. ACM, New York, pp 44–52. http://doi.acm.org/10.1145/1284420.1284435
5. Cohn R (1993) Portable document format reference manual. Addison-Wesley Longman Publishing Co, Inc, Boston
6. Cold SJ (2006) Using really simple syndication (rss) to enhance student research. SIGITE Newsl 3(1):6–9. http://doi.acm.org/10.1145/1113378.1113379
7. Flanagan D (2006) JavaScript: The definitive guide. O'Reilly Media, Inc
8. Fleischer M (1995) Simulated annealing: past, present, and future. In: WSC'95: Proceedings of the 27th conference on winter simulation. IEEE Comput Soc, Washington, pp 155–161. http://doi.acm.org/10.1145/224401.224457
9. Giannetti F (2008) An exploratory mapping strategy for web-driven magazines. In: DocEng'08: Proceeding of the eighth ACM symposium on document engineering. ACM, New York, pp 223–229. http://doi.acm.org/10.1145/1410140.1410188
10. Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley Longman Publishing Co, Inc, Boston
11. Goldenberg E (2002) Automatic layout of variable-content print data. Master's thesis, School of Cognitive & Computing Sciences, University of Sussex, Brighton, UK. http://www.hpl.hp.com/techreports/2002/HPL-2002-286.html
12. Harrington SJ, Naveda JF, Jones RP, Roetling P, Thakkar N (2004) Aesthetic measures for automated document layout. In: DocEng'04: Proceedings of the 2004 ACM symposium on document engineering. ACM, New York, pp 109–111. http://doi.acm.org/10.1145/1030397.1030419
13. Hurst N, Li W, Marriott K (2009) Review of automatic document formatting. In: DocEng'09: Proceedings of the 9th ACM symposium on document engineering. ACM, New York, pp 99–108. http://doi.acm.org/10.1145/1600193.1600217
14. Jacobs C, Li W, Schrier E, Bargeron D, Salesin D (2003) Adaptive grid-based document layout. In: SIGGRAPH'03: ACM SIGGRAPH 2003. ACM, New York, pp 838–847
15. Johari R, Marks J, Partovi A, Shieber S (1997) Automatic yellow-pages pagination and layout. http://citeseer.ist.psu.edu/johari97automatic.html
16. Lin X (2005) Active document layout synthesis. In: Proceedings, eighth international conference on document analysis and recognition, 29 Aug–1 Sept 2005, vol 1, pp 86–90. doi:10.1109/ICDAR.2005.42
17. Lumley J, Gimson R, Rees O (2005) A framework for structure, layout & function in documents. In: DocEng'05: Proceedings of the 2005 ACM symposium on document engineering. ACM, New York, pp 32–41. http://doi.acm.org/10.1145/1096601.1096615
18. Morrison M, Brownell D, Boumphrey F (1999) Xml unleashed. Sams, Indianapolis
19. de Oliveira JBS (2009) Two algorithms for automatic page layout and possible applications. Multimed Tools Appl 43(3):275–301. doi:10.1007/s11042-009-0267-y
20. Plass MF (1981) Optimal pagination techniques for automatic typesetting systems. PhD thesis, Stanford, CA, USA

21. Purvis L, Harrington S, O'Sullivan B, Freuder EC (2003) Creating personalized documents: an optimization approach. In: DocEng'03: Proceedings of the 2003 ACM symposium on document engineering. ACM, New York, pp 68–77. http://doi.acm.org/10.1145/958220.958234
22. Schrier E, Dontcheva M, Jacobs C, Wade G, Salesin D (2008) Adaptive layout for dynamically aggregated documents. In: IUI'08: Proceedings of the 13th international conference on intelligent user interfaces. ACM, New York, pp 99–108. http://doi.acm.org/10.1145/1378773.1378787
23. Sellman R (2007) Vdp templates with theme-driven layer variants. In: DocEng'07: Proceedings of the 2007 ACM symposium on document engineering. ACM, New York, pp 53–55. http://doi.acm.org/10.1145/1284420.1284436
24. Skiena SS (1998) The algorithm design manual. Springer, New York