SandSlide: Automatic Slideshow Normalization

Sieben Bocklandt* (✉)[0000−0003−3833−624X]
Gust Verbruggen*[0000−0001−9182−597X]
Thomas Winters[0000−0001−7494−2453]

Department of Computer Science; Leuven.AI
KU Leuven, Belgium
firstname.lastname@kuleuven.be

Abstract. Slideshows are a popular tool for presenting information in a structured and attractive manner. There exists a wide range of different slideshows editors, often with their own proprietary encoding that is incompatible with other editors. Merging slideshows from different editors and making the slide design consistent is a nontrivial and time-intensive task. We introduce SandSlide, the first system for automatically normalizing a deck of slides from a PDF file into an editable PowerPoint file that adheres to the default slide templates, and is thus able to fully leverage the flexible layout capabilities of modern slideshow editors. SandSlide achieves this by labeling objects, using a qualitative representation to find the most similar slide layout and aligning content from the slide with this layout. To evaluate SandSlide, we collected and annotated slides from different slideshows. Our experiments show that a greedy search is able to obtain high responsiveness on supported and almost supported slides, and that a significant majority of slides fall into this category. Additionally, our annotated dataset contains fine-grained annotations on different properties of slideshows to further incentivize research on all aspects of the problem of slide normalization.

Keywords: Slideshow normalization, Document annotation

1 Introduction

Slideshows are one of the most popular methods for transferring information in a structured and attractive manner to an audience. Many different applications allow users to easily create their own slideshows, the most popular of which are *PowerPoint*, *KeyNote*, *Google Slides*, *Beamer* and *Prezi*. Each of these applications has the same goal of helping a user to design a beautiful deck of slides with text, images and other multimedia content. In order to do so, they provide design templates that control the look and feel of all slides. These design templates allow the user to focus on their content, rather than requiring them to spend hours overlooking the individual layout of each slide.

One large issue is that most applications use their own proprietary encoding format and compatibility between these tools is therefore extremely low. For

* Sieben and Gust contributed equally to this work.

example, it is borderline impossible to edit a given *KeyNote* presentation in *PowerPoint* and vice versa. Merging slides from different editors into a single slideshow with a uniform design requires the user to manually fix each slide, often making it easier to create the uniform slides from scratch. Even the simple task of presenting a slideshow on a machine that does not have the correct application will often cause it not to be displayed correctly.

To address this last issue, there is one type of file that almost all editors can export slides to: the Portable Document Format (PDF). While sacrificing some of the functionality provided by each tool, this encoding allows for slideshows to be opened on platforms that do not have the slideshow editor itself. The main drawback of exporting a slideshow as a PDF is losing the ability to edit the slideshow in a slide editor. Several tools are available for converting PDF slideshows back into editable slideshows, but they tend to bluntly convert the content without any regard for the templates that slides were initially designed with. The user can then edit text or replace images, but they cannot easily change the overall design of the slideshow. For example, the font of each piece of text is attached to that specific piece of text, as opposed to having a uniform font for the whole slideshow as provided by the design template.

We propose a new normalization process for converting PDF slideshows back into an editable slideshow that respects the templates provided by slideshow editors. In the resulting slideshow, each object is assigned a specific role using placeholders and the editor decides where it places objects based on this role and the design template that the user selected. Examples of such roles could be *title* or *leftmost column of a slide with multiple columns*. Changing the template then changes the overall design of the slideshow. This allows interoperability between different applications, as it allows different slideshows to be merged seamlessly or the design of different slideshows to be made uniform with minimal effort.

As the exported PDF does not contain semantic information about the objects it contains, nor where they came from in the original editor, this task is highly nontrivial. Even worse, the words that make up a single paragraph are often split up as different objects. The normalization process thus needs to discover the high-level objects that make up a slide, assign them a semantic label and align these labeled objects with those of one of the slide layouts to discover their role.

In this paper, we make the following contributions:

- We identify and introduce the problem of normalizing slides by assigning a specific role to each object.
- We describe and implement an algorithm based on qualitative representations of slides to obtain this assignment of objects to roles.
- We create and release a benchmark dataset to evaluate the novel problem of slide normalization.

2 Background

We start by providing background on the structure of slideshows and describing the limitations of existing converters.

2.1 Slideshow Structure

A slideshow is a sequential set of slides filled with objects. All slides generally follow the same *design template* that specifies the default fonts, colors, backgrounds and style of other components for each type of slide layout. Each slide is instantiated using a particular slide layout, which defines the objects that it contains and their roles. These roles are implicit and the user interacts with them by editing *placeholders* for the objects of each role. A placeholder is thus an object that the user interacts with that has an implicit role defined by the slide layout.

Example 1. Examples of slide layouts are *Title slide*, *Section header* and *Title with double content*. The latter of these has three placeholders, one for the title, one for its left content and one for its right content.

Most design templates provided by editors support the same slide layouts. When changing to a different design template, objects in placeholders automatically move to the positions defined by their role in the corresponding layout of the new template. Objects not in a placeholder remain in the same position.

Example 2. Figure 1 shows three slide layouts in the default design template and their placeholders. The user can edit these placeholders and fill them with content. By selecting a new design template, all slides are automatically adapted to the layouts defined by this new template.

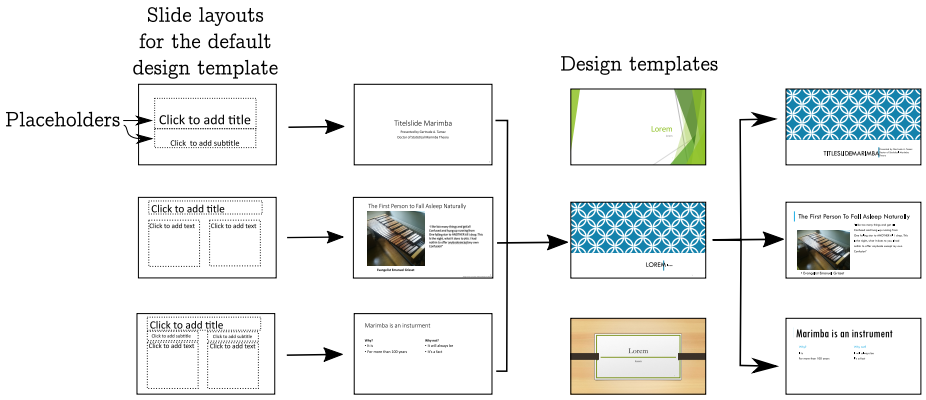


Fig. 1: Steps a user encounters when creating a new slideshow in PowerPoint. Slides are created by filling placeholders in slide templates and the layout is decided by choosing a specific slide master.

2.2 PDF to PPT converters

Several online tools for converting PDF files into editable PowerPoint files already exist, such as `smallpdf`, `ilovepdf`, `pdf2go`, `simplypdf` and `online2pdf`.* For each page in the PDF, these applications first create a slide with the empty slide layout and then convert all objects of the PDF page into PowerPoint objects. Not only do they not use placeholders, some applications even convert text into images, making it even harder to change the design of the slides. The inability of dealing with a change of design template is illustrated in Figure 2.

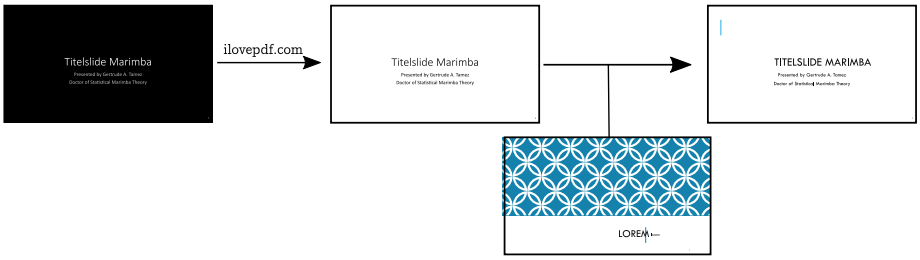


Fig. 2: A slide converted with `ilovepdf.com` does not adapt its layout when changing the new design template.

2.3 Automatic Document Annotation

Automatic document annotation enables the retrieval of structured textual content from PDF files. In this field, algorithms are crafted or trained to automatically labeling parts of the document with their function, such as *abstract*, *title* and *author*. These annotations are then used to improve the performance of further downstream tasks, such as search. While most document annotation tools use heuristics for annotating the text [5, 10], some also use limited machine learning methods—such as support vector machines—to additionally annotate metadata [11]. To provide the required data to train such document annotation algorithms, researchers recently released PubLayNet, a dataset containing multiple hundreds of thousands annotated documents [14]. While document annotation tools usually focus on annotating formal documents, the task of slide segmentation has recently been gaining attention. Slide segmentation tools like SPaSe and WiSe are build to automatically segmenting pictures of slides by detecting regions using neural architectures [6, 7].

* Found at their `name.com`.

3 Slideshow Normalisation

In this section, we describe the problem of slide normalization and define the scope of this paper.

3.1 Problem Statement

The goal of normalizing a slideshow is to allow users to easily change its design by editing or changing the design template. This is trivial when all content is added in a placeholder and thus has a semantic role. We therefore define a *normal slide* as a slide in which all objects are stored in placeholders. A *normal slideshow* consists of only normal slides.

Given a slide and a set of design templates, the problem of slide normalization is to select one of these templates such that each object of the slide is assigned to exactly one placeholder and each placeholder is assigned exactly one object in such a way that objects from the input slide fulfill the role in the normalized slide that the user intended them to. Evaluating whether an object is assigned *the correct* role is impossible without human intervention. It is the task of the normalization algorithm to try and maximize the probability of this happening.

Example 3. Three examples of slides and their normalization in the default design template are shown in Figure 3. It is important that *"Lorem"* is assigned to the placeholder which fulfills the title role and not to the subtitle placeholder.

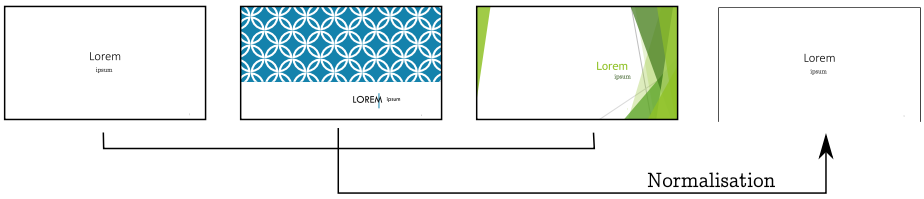


Fig. 3: Three slides that share the same normalization

3.2 Scope

In this paper, we limit the scope of object types considered in the normalization process. Our system currently deals with text objects (normal text, bullet lists, captions, footers, slide numbers) and pictures. More complicated objects—such as SmartArt, mathematical equations and charts—are thus out of scope for this paper. Some of these are easy to integrate with the correct preprocessing, such as charts, and others are interesting pointers for future work, such as arrows and equations.

4 SandSlide: Automatic Slideshow Normalization

We propose and implement a method called SandSlide (*Searching to Automatically Normalize Decks of Slides*) for automatically obtaining normal slideshows from PDF exports. SandSlide is based on three main components. First, it detects high-level objects and annotates them with a semantic label. Second, it uses spatial relations between these objects to obtain a qualitative representation of a slide, which allows for aligning the objects across two slides and quantitatively comparing them. Third, it searches for the best alignment between the objects of a slide and a layout. The obtained alignment can be used to convert the slide into its normalized equivalent.

4.1 Detecting and Labeling Objects in Slides

In the first step, SandSlide looks for and semantically labels objects from the PDF slide. This is achieved in four steps, which are illustrated in Figure 4 and briefly covered in the following paragraphs.

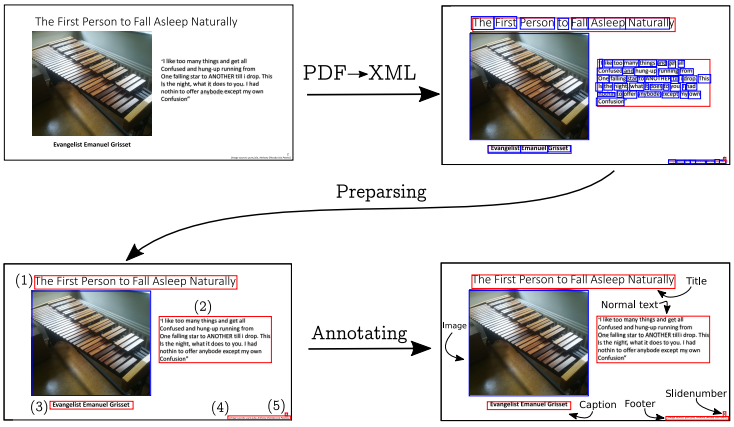


Fig. 4: Converting a page of a PDF file to annotated slide objects.

PDF to objects First, it converts the input PDF file into XML using `pdf2txt`*. SandSlide then removes all objects that are not text or images; either because they are artifacts from the tool, like rectangles around text boxes and single-color pictures, or because they are out of the scope of the system, like curves and SmartArt. The remaining objects are grouped into high-level objects. In Figure 4, this corresponds to the PDF → XML and Preparsing steps.

* `https://github.com/euske/pdfminer/blob/master/tools/pdf2txt.py`

Annotating objects Each of the remaining objects is then assigned a label representing its most likely type of content. Heuristic approaches are often used in document annotation methods and have as the main benefit that they do not require training data [5, 10, 11]. Each object is assigned a score for each label using a local heuristic, computed as the sum of several smaller heuristics. An overview of these heuristics for all labels is shown in Table 1. The label assigned to an object is that with the highest score. The *mean title* is the weighted mean of the bounding boxes of the title in previous slides, with the slide number as weights.

Example 4. The normalized feature vectors for the text objects of the slide in Figure 4 are shown in Table 2. The assigned labels are printed in bold.

Table 1: Local heuristics for assigning a label to objects in a slide, and whether or not it is used in the qualitative representation later (denoted by Q). Bold words are used to refer to objects in Table 2.

Label	Heuristic	Weight	Q
background	Area covers more than 80% of the slide.	1.0	No
listing image	Two or more identical images are placed in a vertical line.	1.0	No
normal image	None of the above.	1.0	Yes
slide **number**	Positioned on the outside 20% of the slide.	0.33	No
	Follows `\d+` or `\d+[/\]\d+`.	0.66	
title	Largest font size in slide.	0.14	Yes
	Positioned in the upper 40% of the slide.	0.29	
	More than 20% overlap between object and *mean title*.	0.57	
	First slide and largest font size.	1.0	
	Largest font size is similar to font size of title in first slide	0.57	
	Not in upper 40% of the slide.	0.43	
footer	Positioned in the outside 10% of the slide	0.33	No
	Either contains a word from {src, source,...} or is a URL.	0.66	
listing	Over 20% of the sentences start with a listing character or listing image.	0.66	Yes
	Contains more than one line of text.	0.33	
caption	Image is positioned 20% of the slide height above the object with a 90% horizontal overlap.	0.75	No
	Is a single line of text.	0.25	
normal text	None of the above.	0.5	Yes

4.2 Qualitative Representation

Annotated slide objects are now converted into a qualitative representation. Such a representation provides a level of abstraction on top of the numerical properties

Table 2: Feature vectors for the example slide in Fig. 4

object	number	title	footer	listing	caption	normal
1	0.00	**0.53**	0.00	0.00	0.16	0.31
2	0.00	0.00	0.00	0.40	0.00	**0.60**
3	0.00	0.00	0.00	0.00	**0.67**	0.33
4	0.00	0.00	**0.57**	0.00	0.14	0.29
5	**0.48**	0.00	0.16	0.00	0.12	0.24

of objects and is typically used in systems that need to reason about concepts in space and/or time [9]. In the context of this paper, it is hard to compare a slide with a template based on the exact positions of their objects. Simply knowing whether an object is placed *above* or *left of* another object then provides enough information to compare two slides and align their objects.

The Allen relations [1] describe seven configurations of two intervals, as illustrated in Figure 5a. By projecting two objects on the x- and y- axes defined by the top and left border of a slide, their relative position can be uniquely described by exactly two of these relations, one in each dimension. We write $r_x(a, b)$ if relation r holds between objects a and b along the x-axis.

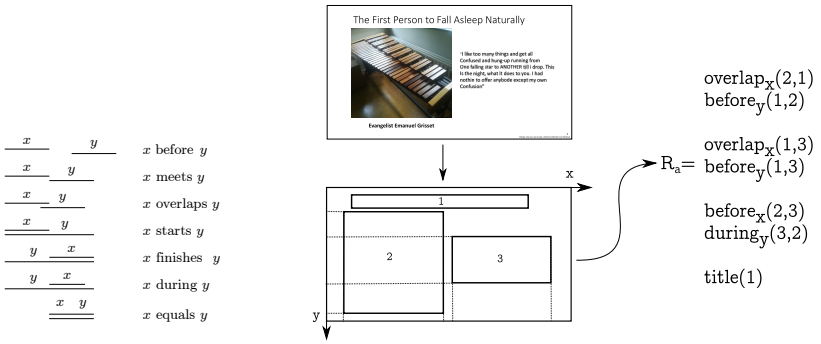
Example 5. In Figure 4, we can see that $\text{before}_x(2, 3)$ and $\text{during}_y(3, 2)$ hold.

The qualitative representation of a slide is obtained by the Allen relations between all pairs of objects that are not trivially aligned. Slide numbers and footers, for example, are trivially aligned across two slides. The Q column in Table 1 indicates for every object whether it is included in the qualitative representation or not. Additionally, we extend this set of relations with the unary $\text{title}(o)$ and $\text{background}(o)$ predicates and whether or not the slide is the first slide of the slideshow. The title serves as an anchor for comparing two slides, as it is present in all slide layouts that contain other content. A full example of the qualitative representation of a slide is shown in Figure 5b. We write R_a to describe the representation of a slide a.

4.3 Searching for Slide Layouts

The final and most important step is finding the most suitable slide layout for each slide. This is achieved by first creating possible reference slides for each slide layout and then (qualitatively) moving objects in the given slide until its representation is equal to that of one of the reference slides. This process is represented schematically in Figure 6.

Generating Reference Slices For each of the given slide layouts, SandSlide generates reference slides by filling their placeholders with content of varying size and selecting different design templates. Each reference slide is converted to a qualitative representation using the method described in the previous section.



(a) Allen relations (b) Qualitative representation using the Allen relations

Fig. 5: Allen relations are used to obtain a qualitative representation of a slide.

Obtaining more reference slides can be interpreted as a *backward search* that yields more target slides to be found at the cost of requiring more comparisons after each moved object.

Qualitatively Comparing Slides Two slides are equal if their representation is equal, but this is not invariant to a permutation of the identifiers for objects in slides. For example, two slides with representations

$$R_1 = \{\text{before}_x(1,2), \text{during}_y(1,2)\} \quad \text{and} \quad R_2 = \{\text{before}_x(2,1), \text{during}_y(2,1)\}$$

are equal, but their representations are not. A *substitution* is a transformation of a representation that simultaneously replaces an object identifier with another identifier in all of its relations. For example, the substitution $\{1 \to 2, 2 \to 1\}$ can be applied to either R_1 or R_2 to make them equal. More generally, when comparing two slides a and b, we say that they are equal if and only if there exists a substitution θ such that $\theta(R_a) = R_b$.

Finding such a substitution is called the *set unification problem* and is shown to be NP-complete [8]. Many algorithms for set unification have been presented and exact details are considered out of the scope for this paper. Anchoring the title across two slides allows us to greatly prune the search space and we use a brute force approach.

Qualitatively Transforming Slides Rarely will an arbitrary slide be equal to one of the reference slides. SandSlide will thus perform qualitative transformations of the representation of the given slide and match the transformed slide with references. Let R be the set of Allen relations. A *transformation* t is a function that replaces one or more relations $r_d(o_1, o_2) \in R$ with a new relation $r'_d(o_1, o_2) \in I$ along the same axis d. The length of a transformation $|t|$ is the number of replacements that it performs.

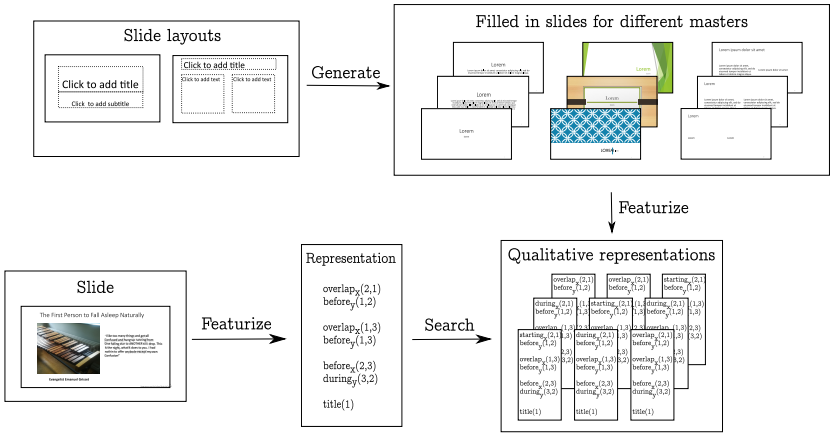


Fig. 6: Discovering the most suitable slide layout type for a slide based on its qualitative representation.

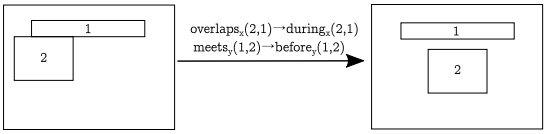


Fig. 7: Qualitative transformation of objects.

Fig. 8: Confusion matrix of annotations with local heuristics.

Example 6. An example of a transformation of length 2 is shown in Figure 7.

Not all transformations yield a qualitative representation for which there exists an actual configuration of objects. Checking if such a configuration exists is called the *satisfaction problem for interval algebra* and it is also shown to be NP-complete [12]. We opt to simply not check whether a transformed representation is consistent as these intermediate, inconsistent representations serve as stepping stones to find an exact match.

Searching for Slides Given a slide s and a set of slide layouts $\mathbf{L}$ with each layout $L_i \in \mathbf{L}$ described by a set of representations $l_i^j \in L_i$, we then look for the smallest transformation t such that there exists a representation $l_i^j \in L_i$ such that $t(s)$ is equal to l_i^j.

A slide with n objects is represented by $2n$ relations. Each relation can be transformed into six new relations. There are then $\binom{n(n-1)}{d} \times 6^d$ different transformations of length d. Each transformed slide has to be compared with all reference slides. Even for small d, this quickly becomes intractable to compute.*

We therefore use a heuristic algorithm based on the similarity between a slide and all reference slides. The Jaccard similarity $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$ is a common way to compute the similarity between two sets A and B. In order to obtain a similarity between two slides a and b that respects an optimal alignment between objects, we compute

$$sim(a,b) = \max_{\theta \in \Theta} J(\theta(a), b)$$

with Θ all possible substitutions. The score for a transformation on the original slide is

$$score(t) = \max_{L \in \mathbf{L}} \max_{l \in L} sim(l, t(\text{slide})) \tag{1}$$

with $\mathbf{L}$ the set of all layouts, L a specific layout, l a possible representation of that layout and $t(\text{slide})$ the representation of the transformed slide. Note that the similarity for equal slides is 1 and computing the heuristic also informs us when an exact match is found. We do not need to find the optimal substitution twice as the heuristic is obtained when checking for a solution.

SandSlide performs a greedy search using the heuristic. At each step, the best transformation so far according to Equation 1 is expanded. All six possible transformations of a relation are considered at once. Relations across different axes are transformed independently. Loops are prevented by first generating all transformations of length one and combining these to form larger transformations.

If a solution is not found after a few transformations, any solution is not likely to be closely related to the intended solution. Search is therefore cut short after a predefined number of comparisons, in other words, the number of times an optimal substitution has to be computed. The layout assigned to a slide is then given by

$$solution(s, \mathbf{L}) = \arg\max_{L_i \in \mathbf{L}} \max_{l \in L_i} \max_{t \in T} sim(l, t(s)) \tag{2}$$

where T is the set of all transformations evaluated during search.

5 Evaluation

We performed several experiments to answer the following research questions.

Q1 Do we require search or can we simply look for the most similar reference slide according to the heuristic?
Q2 What is the effect of performing a deeper backward search at the cost of requiring more comparisons in each iteration?
Q3 Does our algorithm guarantee that objects are assigned their intended role?

* For a slide with five objects, there are 25920 transformations of length 3. For as little as 40 reference slides, that is over a million comparisons.

5.1 Experimental Setup

Data In order to answer these questions, we start from a set of 1000 slideshows provided by the U.S. Library of Congress [3]. We filtered out all files that were not PowerPoint files and slideshows that are completely out of scope for slideshow normalization, which leaves 640 slideshows.

From those slideshows, we manually annotated a randomly sampled set of 500 slides using VIA [4] after the PDF → XML and the preparsing steps. Two types of annotations were recorded; the slide as a whole and individual objects. For slides, we make the following annotations; (1) objects in scope and corresponds to a layout; (2) objects in scope and superset of a layout; (3) objects not in scope, but would otherwise fit 1 or 2; (4) does not match a layout but would make an interesting layout or (5) completely out of scope. Slides of type (1) and (2) are used in our evaluation. For objects, we made precise annotations of their semantic type and their intended role. We use these last annotations as a human evaluation of the algorithm, as it allows to check if the annotations assigned by the algorithm match those assigned by humans.

The role of an object denotes the positions on a slide where it is allowed to end up after transformation. For example: when there are two images besides each other, the one on the left should stay on the left or when there is a title and a subtitle, the subtitle should be used as a subtitle and not as content. This implies that the image on the left should never be the rightmost piece of content and that the subtitle should be below the title and above the content.

These annotations are more precise than required for our experiments. Our goal is to encourage research on different aspects of slideshow normalization and to work towards an established benchmark for this problem. We make the code, original slides, results after the preparsing and annotation steps and all ground truth annotations publicly available.⋆⋆

Evaluation We measure two properties of aligning a slide with a reference. The *responsiveness* is the proportion of objects that can be assigned to a placeholder. A random assignment of objects to placeholders trivially yields high responsiveness. We therefore use annotated roles to compute the *sensibility* of an alignment as the proportion of alignments that are allowed with a set of handcrafted rules. The sensibility measure acts as qualitative evaluation, as it checks if all the elements stay in allowed positions.

We do not compare SandSlide with existing online tools (2.2) using these measures as the results are trivial: they do not assign objects to placeholders and place the objects on the same position as in the PDF, which results in zero responsiveness and undefined sensibility.

Reference slides Most design templates support ten layouts. Using these layouts, we created three levels of reference slides. The **baseline** set considers

⋆ `https://github.com/zevereir/SandSlide_data`

⋆ `https://github.com/zevereir/SandSlide`

only the placeholders from the default design theme, without filling them with content, resulting in one reference for each layout. The **learned** set considers the default design theme, where placeholders are filled in with content of different lengths. This results in 137 representations. The **masters** set also considers different design themes, which results in 1117 unique representations. Limiting the search based on the number of comparisons was done to ensure a fair comparison between the sets of references.

5.2 Results

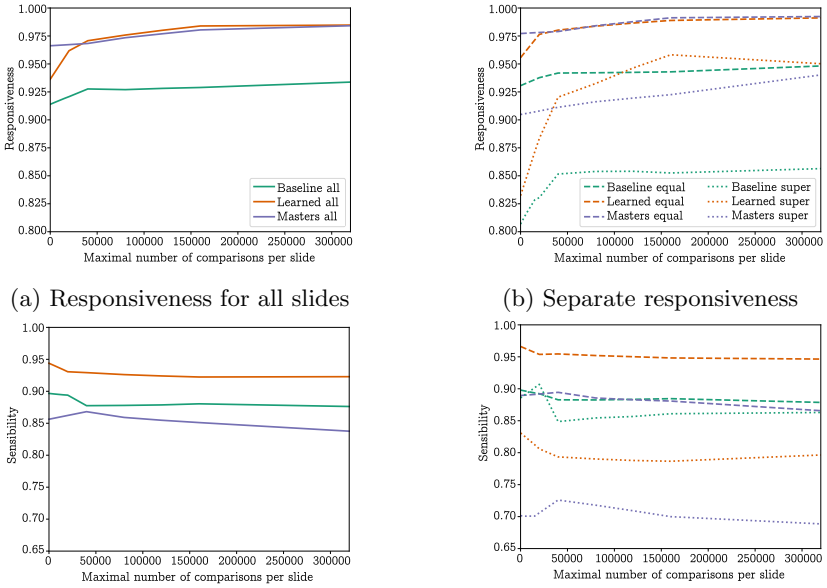
We did not perform an extensive evaluation of object annotation using local heuristics, as we found them to be sufficiently accurate for our purposes. A confusion matrix of assignments is shown in Figure 8. Each row represents the normalized number of assignments made by the local heuristics. Listings, normal text and pictures are all content and no distinction is made between them in the qualitative representation. Some objects are not used in the representation, either because they are trivially aligned or no placeholder exists for them. Our experiments focus on evaluating the qualitative representation and search.

Responsiveness and sensibility for all sets of reference slides are plotted in function of the maximal number of comparisons in Figure 9. Figure 10 shows the distribution of the slides over the different transformation lengths. We use these figures to answer the research questions in the following paragraphs.

Q1: Requiring search We can see that the heuristic alone obtains respectable results, but that even a little bit of search significantly improves results, especially for the learned set. Searching for longer transformations increases the number of alignments—the responsiveness increases—but that some faulty alignments are introduced, which can be seen from the decreasing sensibility. Greedy search is able to correctly and quickly identify almost all slides with supported layouts.

Q2: Backward search More references yields improved results, with the learned set outperforming the other sets in both methods. Adding references from different designs does not improve responsiveness, as many comparisons are wasted on the excess references. The masters set only catches up in responsiveness after a high number of comparisons, at the cost of significantly more wrong alignments. The effect of the backward search on the transformation lengths is as expected: more possible references causes less transformations needed to complete the search.

Q3: Alignments As expected, deeper searches yield more alignments at the cost of making more mistakes. In general, our greedy method obtains very sensible alignments, even for searches with many comparisons. Adding references from different design templates has a negative effect on the sensitivity as most slides follow classic layouts.



(a) Responsiveness for all slides

(b) Separate responsiveness

(c) Sensibility for all slides

(d) Separate sensibility

Fig. 9: Left: Main results for all slides (`all`) – Right: Main results for slides that exactly match a layout (`equal`) and supersets of layouts (`super`). Each data point is the average over all slides for an experiment with a specific parameter—the maximal number of allowed comparisons per slide.

5.3 Rebuilding slides

From the layout and alignment, SandSlide can build a new slide. An example that obtained perfect responsiveness and sensibility is shown in the top part of Figure 11. The logos on top are copy-pasted onto the new slide, to avoid losing any information, and the slide number is trivially aligned with a placeholder, as can be seen after applying a new template. The bottom of Figure 11 shows an example of an imperfect normalization, as the subtitle in the original slide is converted to left content in the normalized one. The new slide has a perfect responsiveness, but only two out of three elements in the slide were assigned a sensible role.

6 Conclusion and Future work

This paper introduces the nontrivial problem of slideshow normalization and implemented a first system for solving it, called SandSlide. This system is able to transform a given PDF file representing a slideshow into a normalized slideshow with a responsive slide layout using placeholders. The resulting slideshow can easily be edited and transformed into another design template. Our experiments

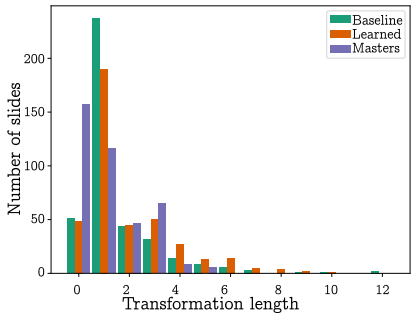


Fig. 10: Distribution of the number of replacements to find the best alignment in the experiment with the learned set and maximal 15000 comparisons per slide.

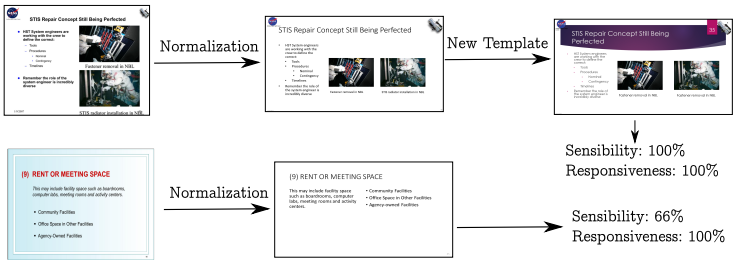


Fig. 11: (top) Perfectly normalized slide with new template. (bottom) Normalized slide with imperfect sensibility as the subtitle is assigned the role of left content.

show that search is a necessary component, that the heuristic works well and that it is beneficial to generate more varied representations for each slide layout, but that too many reduces the quality of the alignment.

Future Work An immediate pointer for follow-up is to support more complex objects—like curves and word art—and objects that require anchors to existing objects—like arrows. This will make the search more complicated and using relational learners [2] would allow to easily incorporate background knowledge about these objects. Our qualitative representation provides an excellent starting point for such learners. As SandSlide allows to efficiently annotate objects and retrieve their semantic role, it can be used to improve downstream tasks involving slideshows. For example, it can be used to create datasets for approaches that learn to automatically generate slideshows [13].

Acknowledgments This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No [694980] SYNTH: Synthesising Inductive Data Models). Thomas Winters is supported by the Research Foundation-Flanders

(FWO-Vlaanderen, 11C7720N). We would like to thank Luc De Raedt for his frustration with merging slideshows, which led to the master's thesis that formed the basis for this research.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11), 832–843 (1983)
2. De Raedt, L.: Logical and relational learning. Springer Science & Business Media (2008)
3. Dooley, C.: In the library's web archives: 1,000 u.s. government powerpoint slide decks (2018), https://blogs.loc.gov/thesignal/2019/11/in-the-librarys-web-archives-1000-u-s-government-powerpoint-slide-decks/, last visited 04/02/2021
4. Dutta, A., Zisserman, A.: The VIA annotation software for images, audio and video. In: Proceedings of the 27th ACM International Conference on Multimedia. MM '19, ACM, New York, NY, USA (2019), https://doi.org/10.1145/3343031.3350535
5. Ferrés, D., Saggion, H., Ronzano, F., Bravo, A.: PDFdigest: an Adaptable Layout-Aware PDF-to-XML Textual Content Extractor for Scientific Articles. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018). European Language Resources Association (ELRA), Miyazaki, Japan (May 2018), https://www.aclweb.org/anthology/L18-1298
6. Haurilet, M., Al-Halah, Z., Stiefelhagen, R.: Spase-multi-label page segmentation for presentation slides. In: 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 726–734. IEEE (2019)
7. Haurilet, M., Roitberg, A., Martinez, M., Stiefelhagen, R.: Wise—slide segmentation in the wild. In: 2019 International Conference on Document Analysis and Recognition (ICDAR). pp. 343–348. IEEE (2019)
8. Kapur, D., Narendran, P.: Np-completeness of the set unification and matching problems. In: International conference on automated deduction. pp. 489–495. Springer (1986)
9. Kuipers, B.: Qualitative reasoning: Modeling and simulation with incomplete knowledge. Automatica **25**(4), 571–585 (1989), https://www.sciencedirect.com/science/article/pii/000510988990099X
10. Luong, M.T., Nguyen, T.D., Kan, M.Y.: Logical structure recovery in scholarly articles with rich document features. Int. J. Digit. Libr. Syst. **1**, 1–23 (2010)
11. Tkaczyk, D., Szostek, P., Fedoryszak, M., Dendek, P.J., Bolikowski, L.: CERMINE: automatic extraction of structured metadata from scientific literature. International Journal on Document Analysis and Recognition (IJDAR) **18**(4), 317–335 (Dec 2015), https://doi.org/10.1007/s10032-015-0249-8
12. Vilain, M., Kautz, H., Van Beek, P.: Constraint propagation algorithms for temporal reasoning: A revised report. In: Readings in qualitative reasoning about physical systems, pp. 373–381. Elsevier (1990)
13. Winters, T., Mathewson, K.W.: Automatically generating engaging presentation slide decks. In: Computational Intelligence in Music, Sound, Art and Design - 8th International Conference, EvoMUSART. Ekart, A., Lecture Notes in Computer Science. Springer, Cham (2019)
14. Zhong, X., Tang, J., Yepes, A.J.: Publaynet: largest dataset ever for document layout analysis. In: 2019 International Conference on Document Analysis and Recognition (ICDAR). pp. 1015–1022. IEEE (2019)