VaryLaTeX: Learning Paper Variants That Meet Constraints
Mathieu Acher, Paul Temple, Jean-Marc Jézéquel, José Ángel Galindo Duarte, Jabier Martinez, Tewfik Ziadi

To cite this version:

HAL Id: hal-01659161
https://inria.hal.science/hal-01659161
Submitted on 12 Dec 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Vary\LaTeX{}: Learning Paper Variants That Meet Constraints

Mathieu Acher
Paul Temple
Jean-Marc Jézéquel
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
mathieu.acher@irisa.fr

José A. Galindo
University of Sevilla
Sevilla, Spain
jagalindo@us.es

Jabier Martinez
Tewfik Ziadi
Sorbonne University UPMC
Paris, France
jabier.martinez@lip6.fr

ABSTRACT
How to submit a research paper, a technical report, a grant proposal, or a curriculum vitae that respect imposed constraints such as formatting instructions and page limits? It is a challenging task, especially when coping with time pressure. In this work, we present Vary\LaTeX{}, a solution based on variability, constraint programming, and machine learning techniques for documents written in \LaTeX{} to meet constraints and deliver on time. Users simply have to annotate \LaTeX{} source files with variability information, e.g., (de)activating portions of text, tuning figures’ sizes, or tweaking line spacing. Then, a fully automated procedure learns constraints among Boolean and numerical values for avoiding non-acceptable paper variants, and finally, users can further configure their papers (e.g., aesthetic considerations) or pick a (random) paper variant that meets constraints, e.g., page limits. We describe our implementation and report the results of two experiences with Vary\LaTeX{}.

KEYWORDS
\LaTeX{}, technical writing, machine learning, constraint programming, variability modelling, generators

1 INTRODUCTION
Technical writing is the process of producing technical or scientific documents such as articles, books, theses, technical reports, deliverables or grant proposals. Besides filling the content of the document, writers should usually comply with 1) precise formatting instructions (e.g., page limit and document templates) and 2) strict deadlines to submit the final document. When the deadline is approaching, a recurrent, challenging and stressful task for writers is to meet the constraints in formatting instructions and page limits to deliver the document on time. As a result, writers usually apply a quick-and-dirty\textsuperscript{1} solution that consists in editing the content, layout, or styles of the document. Writers can make figures smaller, tweak line spacing, use abbreviations, apply a different style (e.g., font size) for a portion of text, remove apparently superfluous text or references, or other techniques to avoid missing the deadline that will make useless all the writing effort\textsuperscript{2}.

Of course, the best strategy is to properly revise the whole document (e.g., carefully identifying unnecessary content). However, in practice, this ideal solution is neither always possible (deadline might be imminent) nor sufficient (e.g., the page limit is hard to fulfill). The challenge is especially more complex when using\textsuperscript{3}.

\textsuperscript{1}According to Oxford dictionary, this adjective is used for describing a quick calculation, method, etc., especially one that is done or used until you have enough time or money to do or use a more careful one.

\textsuperscript{2}A PhD comic nicely illustrates the situation: \url{http://phdcomics.com/comics.php?f=1971}

\textsuperscript{3}An automatic process using machine learning to learn how to meet the constraints using the annotated \LaTeX{} source.

\textsuperscript{4}We report on two real usages of Vary\LaTeX{} for a scientific paper and for a curriculum vitae with their corresponding constraints.

\textsuperscript{5}We make Vary\LaTeX{} source code and experiments’ data publicly available.

\url{https://github.com/FAMILIAR-project/varylatex}
This paper is structured as follows: Section 2 presents Vary\LaTeX \ for generating and learning paper variants that meet constraints. Section 3 describes our implementation. Section 4 reports two experiences. Section 5 presents related work. Finally, Section 6 presents the conclusions and outlines future work.

2 LEARNING CONSTRAINTS TO MEET PAPER VARIANTS’ CONSTRAINTS

Our solution is based on variability, constraint programming, and machine learning techniques. Figure 1 describes the different steps of the process that we detail in this section.

Variability annotation and modeling. First, users have to annotate \LaTeX source files with variability information, e.g., (de)activating portions of text, tuning the figures’ sizes, or tweaking line spacing. We show examples on the left side of Figure 1 and in Figure 2.

Our solution relies on a basic template engine supporting conditional directives and value substitutions. For instance, when we want to include the acknowledgments text that we made optional) and \textbf{BOLD_ACK} is set to true (\textbf{ACK} is optional) and \textbf{BOLD_ACK} is a child of \textbf{ACK}.

\textbf{Acknowledgment.} We thank anonymous re...

Numerical values can also be used. If \texttt{bref\_size} is a percentage of the size of a figure and we set it to 0.8, the corresponding \LaTeX will contain:

\includegraphics[width=0.8\linewidth]...

We show examples on the left side of Figure 1 and in Figure 2. In the right-hand side of Figure 2a, we depict some paper variants corresponding to some combinations of values. We can notice some aesthetics’ differences and that some paper variants take more space.

Variability annotations do not specify the possible values of variables. Users can define the domain values of numerical variables in a dedicated variability model (shown on the left side of Figure 1, below \LaTeX source files). For example, \texttt{bref\_size} can take real values between 0.7 and 1.0 while \texttt{vspace\_bib} can take real values between 1.0 and 5.0. Specific constraints can also be added, as the logical implication between \textbf{BOLD\_ACK} and \textbf{ACK} (in the example, \textbf{BOLD\_ACK} is a child of \textbf{ACK}). Overall, users can encode variability information into an attributed feature model [3].

Paper variants generation. The second step of our process is the automated derivation of papers (thanks to the variability annotations and modeling step). The variability model can be used to pilot the production of outputs (e.g., PDFs) from its valid configurations. We consider that a configuration is an assignment of values to configuration options (i.e., in our example, true/false values for Boolean options like \textbf{ACK} and numerical values for \texttt{bref\_size} and \texttt{vspace\_bib}). A configuration is valid if the values fulfill the logical constraints defined in the variability model. Several strategies can be used for assigning values and sampling configurations, such as automatically picking random valid configurations.
valid configuration, we can resolve the variability of the template and obtain plain \LaTeX files. Such files are then compiled to formats like PDF by calling pdflatex and bibtex tools. Finally, an automated procedure can measure interesting properties of a PDF file, such as the number of pages. At the end of this step, we obtain a configuration matrix. In the example of Figure 1 (see right-hand side), each row corresponds to a configuration of a paper variant and consists in 1) configuration values and 2) the obtained number of pages in the last column.

Based on a configuration matrix, we can already identify some paper variants that fulfill page limits. In our example, we consider that the page limit is 4; we could filter out paper variants with 5 pages. It is already useful since we can pick at least one paper variant that meets the constraints. However, we want to be able to choose among all paper variants that meet constraints, not only among those that were part of the random sampling.

One paper variant is not enough: Configuration vs Specialization. The basic reason is that some paper variants may not be that satisfactory. There might be other constraints (such as aesthetics or the importance of some content) that are hard to formally express or measure with an automated procedure. For example, the removal of paragraphs is something that an algorithm should not decide. Users most likely want to configure every aspect of their papers. In a random paper variant that meets the page limits, perhaps a paragraph or a reference can be kept after all; the size of a figure could be augmented, or the tweaking of line spacing is not really necessary. We face this need in real-world cases, as reported in Section 4. In other words, we pre-configure (specialize) the \LaTeX generator as follows: All options values presented to users should lead to paper variants that meet page limits. On the left side of Figure 1, at the bottom, we depict a specialized configurator. Users can interactively select or deselect Boolean options (with checkboxes) and set values (with sliders). The configurator assists users at each step, typically through the propagation of choices. In our example, we can notice that some configurations (and therefore
paper variants) are now no longer reachable. For instance, when selecting JS_SCRIPTSIZE, users are forced to set a value between 0.6 and 0.65 for cserver size, and PARAGRAPH_ACK is not selectable (intuitively, the use of a paragraph takes too much space).

A Learning Problem. To avoid non-acceptable paper variants, we need to learn constraints among Boolean and numerical options. Setting a specific value to an option is a special kind of constraint. In general, we also want to avoid some combinations of values. In our example, a configuration is considered as acceptable whenever the number of pages equals to 4. Thanks to the configuration matrix, we can classify a sample of configurations as acceptable or non-acceptable. We then address a statistical binary classification problem in which we can predict which options (part of the configurations) lead to acceptable or non-acceptable configurations.

We consider all options of configurations as predictor variables. We have a training sample of \( N \) configurations on \( Y \) class variables that take values \{acceptable, non-acceptable\}. Given training data (see the matrix on the right side of Figure 1), we want to learn a classifier able to predict the values of \( Y \) from new, previously unseen configuration values.

Sampling and Learning. As for numerous existing configurable systems, a major issue is that it is unfeasible to evaluate all configurations (here because of the href_size and vspace_bib that lead to an explosion of possible numerical values). In general, a solution based on a comprehensive enumeration is not possible because of the cost of derivation and measurement. In our case, compiling one \( \LaTeX \) variant can take several seconds. Therefore, in practice, it is more realistic to assume that we rather have at our disposal a (small) sample of configurations.

Many approaches have been developed for addressing (binary) classification problems. We selected classification trees, a supervised machine learning technique that classifies data into classes [10]. Classification trees can handle Boolean and numerical values. Moreover, from classification trees, we can extract human-readable rules or constraints expressed in propositional logics. Specifically, we first build the conjunction of all decisions that lead to leaves that classify configurations as ‘non-acceptable’; we then negate the resulting expression. Such constraints are eventually added in the variability model to exclude non-acceptable configurations. Figure 1 gives an example of a decision tree (in the bottom right corner) and the extracted constraints.

Ready-to-configure. In the final step of our process, users can configure papers that meet page limits. They still can control various options (size of the figure, activation of some content, etc.) and produce a final paper variant. Off-the-shelf solvers internally use the variability model augmented with constraints, typically for propagating choices [6, 8, 18].

3 IMPLEMENTATION

\( \LaTeX \) is implemented in Java and integrated with different specific languages for instrumenting the whole derivation and learning process. As presented in the introduction, Section 1, the source code among exemplary and experimental data is available online. In this section, we go through the different libraries and languages used to develop \( \LaTeX \).

Variability annotations. For annotating \( \LaTeX \) source files, we rely on Mustache\(^4\), a template language. We give some examples of the syntax in Figure 1. It is possible to write conditional directives directly in \( \LaTeX \) (e.g., using \texttt{if} language or off-the-shelf packages). From our experience, the use of an external template language fits our purpose. In particular, we can seamlessly annotate any construct of a \( \LaTeX \) file (e.g., size of a figure or bibliographic reference). For resolving variability within templates, we use Trime\(^5\) a templating engine implementation written in Java. Trime can activate or deactivate some portions of texts or instantiate a specific value within the text.

Variability modeling and sampling. For specifying constraints among configuration options or defining the range of possible values of numerical options, we rely on an extension of FAMILIAR [2]. It allows users to define attributed feature models with a textual language. For reasoning about variability models, we translate them into MiniZinc\(^6\), an open-source constraint modeling language. A MiniZinc model can be imported in a wide range of solvers. We use this translation to generate a sample of valid configurations out of a variability model. We use the Choco solver\(^7\) for randomly choosing some values for each option.

Generating and Learning.\(^\rightarrow\) We generate Bash scripts to 1) instrument the compilation of \( \LaTeX \) source files into PDF; 2) measure each PDF paper; 3) build a CSV file for serializing the configuration matrix. We use R language and the rpart package for producing classification trees out of the CSV. We have also implemented a procedure in Java for extracting constraints out of decision trees.

Usage.\(^\rightarrow\) Our tool expects as inputs annotated \( \LaTeX \) source files and a variability model. The rest of the process is automatic and can be launched through Java. Meanwhile, users can override and control many technical steps of the tool such as the size of the random sampling or the scripts in charge of compiling the files.

4 EXPERIMENTS WITH \( \LaTeX \)

We used \( \LaTeX \) in informal contexts for testing our implementation, prototyping and exploring the potential of the idea. Also, we applied \( \LaTeX \) in two concrete cases.

Finishing a short paper on time. Our first real experience was for finishing the paper [1] on time, both for submitting it and preparing the camera ready. The page limit was 4. Originally the paper was eight pages long and it was rejected in another venue. Despite our effort to rewrite the whole content, we struggled to meet the constraints. After some quick-and-dirty modifications, we found some acceptable paper variants but we were not really satisfied. There were various reasons: The size of some figures tended to be too small; the tweak of line spacing was sometimes too aggressive; and we really wanted to keep some acknowledgments and bibliography references.

There were essentially two related problems. First, our manual trial-and-error process was limited because we only applied individual edits and rarely their combinations. We wanted to explore and cover more cases. Second, we ignored what we should edit and what we could not change at all for meeting page limits. To sum

\(^4\)\url{https://mustache.github.io/}
\(^5\)\url{http://trimou.org}
\(^6\)\url{http://www.minizinc.org/}
\(^7\)\url{http://www.choco-solver.org/}
up, a manual constraint inference based on manual tries and errors had strong limitations. Therefore, we decided to use Vary\LaTeX{} to 1) automate the exploration and 2) learn actual factors that lead to acceptable paper variants.

Variability annotations and modeling. The example in Figure 1 actually corresponds to our case. We used fourteen Boolean options and three numerical options. For instance, vspace_bib was a real option, whose value can range between 1.0 and 5.0. Our variability modeling language supports the specification of precision for real variable i.e., the effective number of decimal digits in it which are treated as significant for computations. The precision here was one decimal digit to limit the combinatorial explosion of possible values. Two other real options were used to affect the width of two figures. We used mutually exclusive Boolean options for specifying the font size


Vary\LaTeX{} guided us in the configuration process and helped to choose an acceptable paper variant.

Sampling paper variants. The number of valid configurations was 73,440. In details, there are 90 valid combinations of Boolean features, 4 possible values for bref_size (either 0.7, 0.8, 0.9, or 1.0), 4 possible values for cserver_size (either 0.6, 0.7, 0.8, or 0.9), and 51 possible values for vspace_bib (either 1.0, 1.1, 1.2, ..., 4.9, or 5.0) in the variability model of Figure 1. As there are no constraints between numerical options and Boolean options, we obtain 90 * 4 * 4 * 51 = 73,440 valid configurations. On average, our experiments showed that the building of a PDF took 4 seconds per configuration. The enumeration of all paper variants would have taken too much time i.e., more than 80 hours. We thus generated a sample of 400 PDFs. In the sample, the proportion of acceptable paper variants was around 52% (i.e., 48% of paper variants had 5 pages).

Needs of learning. Among the acceptable paper variants, we noticed that some of them were not "optimal". For instance, the use of tiny font style for showing snippet codes was considered aggressive; reviewers could hardly read it. The fact is that the activation of this style instantly resolved our problem: all papers were 4 pages. Still, we wanted to investigate whether we could somehow combine other options to use a bigger font size in this case. Overall, it was a tradeoff among the different options regarding aesthetics, readability, and the importance of textual content.

Learning results. When classifying (unseen) configurations and trying to generalize out of a sample, machine learning can introduce mistakes. For example, a paper variant can be classified as acceptable while it is, in fact, non-acceptable. We followed a traditional approach for assessing our learning phase. From the 400 configurations, we randomly selected a certain percentage of configurations for building the training sample; the remaining configurations constituted the testing set. The idea is to first train a classification tree (with a training sample). Then, using the testing set, we can confront the actual classes with the predicted classes (acceptable, non-acceptable).

Results show that we can reach a high accuracy, precision, and recall (more than 85%) even with a low percentage of configurations in the training set (10%). It means that, with the derivation of 40 paper variants, we could already achieve accurate results and learn interesting constraints. We can reach an accuracy, precision, and recall higher than 90% with 200 paper variants in the training set.

Considering the classification trees obtained with 100 configurations in the training set\textsuperscript{6}, we have been able to learn relevant constraints. In particular, we learned we cannot use JS_FOOTNOTESIZE (to keep the example simple, it does not appear at the bottom of Figure 1). Furthermore, we learned we can use another style than JS_TINY (i.e., JS_SCRIPTSIZE) under the condition that we use specific figure sizes or deactivated some content (see the bottom of Figure 1). We finally adopted this solution and fine-tuned the configuration of our paper.

Overall, the experience was positive. First, almost half of papers (out of 73,440) exceeded page limits. Vary\LaTeX{} saved us time and resources. Second, Vary\LaTeX{} guided us in the configuration process and helped to choose an acceptable paper variant.

Sending a CV. Our second real experience was for building a curriculum vitae (CV) that should respect some constraints. Specifically, the CV has to be included as part of a research proposal and should not be longer than 18 pages. We used 5 Boolean options, all options. These options aimed to keep or remove some bibliographic references and some other information about the career. In total, we only had 32 configurations. We used Vary\LaTeX{} for generating all possible paper variants. In this specific case, we did not need the learning phase (as the full exploration of the variability space was possible). Then, we selected one paper out of acceptable paper variants. The choice of the "best" CV variant was a bit arbitrary, but we eventually succeeded to meet constraints.

5 RELATED WORK

We can consider a technical document as an artifact intended for a human to consume. Human-centered product lines have received attention from the research community as a paradigm to adapt Human-Computer Interaction (HCI) artifacts (e.g., user interfaces) to specific constraints, needs or simply to optimize aesthetic or usability preferences \cite{Gears}. We are aware of the substantial difference between HCI artifacts and textual documents. However, there are also commonalities: non functional properties such as readability and aesthetic considerations remain as important factors.

There are tools such as FeatureIDE \cite{pure_variants}, pure::variants \cite{purevariants} and Gears \cite{Gears} that provide variability management functionalities targeting not only source code but also other artifacts including Microsoft Word text documents. Handling this artifact type while sharing most of their core functionality for variability management is possible through an extension in pure::variants \textsuperscript{9} and a bridge in Gears \textsuperscript{10}. These approaches solve the challenge of making effective reuse for documents. However, the derivation of a specific document will not be aware of external constraints (such as page limits) which is one of the purposes of Vary\LaTeX{}.

There are numerous works for predicting the performance of configurations (e.g., [5, 12, 13, 17, 19]). We share the need to learn\textsuperscript{6}\textsuperscript{7}\textsuperscript{8}

\textsuperscript{6}Our observations and results are on average, since we repeated the experiments 10 times.

\textsuperscript{7}pure::variants: https://www.pure-systems.com/products/extensions/pure-variants-connector-for-microsoft-office-311.html

\textsuperscript{8}Gears: http://www.bigleverage.com/ecosystem/bridges/microsoft.html
over a sample of measured configurations. However, our approach boils down to a classification problem while predicting a performance value is a regression problem. Predicting the performance of configurations typically helps to select an optimal configuration. In our case, we aim to synthesize constraints and still allow users to choose among acceptable configurations.

Temple et al. [16] presented an approach that infers constraints from a sample of configurations classified as valid and non-valid. In our case, we compute a quantitative value (number of pages) that is then treated as a categorical variable (e.g., either 4 or 5). It can be used to answer the Boolean question “Do the paper variants meet the formatting instructions?” VaryL builds upon our previous works [14–16] and shows a different applicability of variability and machine learning techniques.

6 CONCLUSION

We presented VaryL, a solution for automatically deriving paper variants based on variability annotations within \LaTeX source files. We described how we can learn constraints among Boolean or numerical values of variation points in such a way that users can only obtain paper variants that meet specific constraints such as page limits. We reported on our experience in developing and applying the tool.

In the short term, we plan to extend our technical implementation for supporting other formats (e.g., Markdown or Word) and improve its usability. We aim to reduce the number of manual annotations by defining a set of predefined cases where adjustments can be usually made (e.g., the vertical separation between a figure and the caption text). We also want to study the state of the art in aesthetic aspects of documents and implement a metric that can be used to qualitatively compare variants of a document. It can eventually discard papers with low aesthetic values. In a sense, we aim to support other end-user constraints than page limits.

As future work, we aim to evaluate whether the variability modeling effort pays off in practice. In general, we believe that our work can be used in several technical writing contexts. We hope to further understand the needs and current practices for eventually expanding the ideas behind VaryL.

ACKNOWLEDGEMENTS

This work benefited from the support of the project ANR-17-CE25-0010-01 VaryVary. This work was supported, in part, by the European Commission (FEDER), by the Spanish government under BELi (TIN2015-70560-R) project, by the Andalusian government under the COPAS (TIC-1867) project and by the ITEA3 15010 REVaMP project: FUI the Île-de-France region and BPI in France.

REFERENCES


