VaryLaTeX: Learning Paper Variants That Meet Constraints

Mathieu Acher, Paul Temple, Jean-Marc Jézéquel, José Ángel Galindo Duarte, Jabier Martinez, Tewfik Ziadi

VaryLaTeX: 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 VaryLaTeX, 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 VaryLaTeX.

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*[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[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 document preparation tools like LaTeXwhich are not based on the WYSIWYG (What You See is What You Get) paradigm like Word or Pages. In LaTeX, authors are encouraged to focus on the content and not in the appearance. Therefore, if authors want to meet the constraints on page limits, they will try some modifications, build the document, verify if the length fulfills page limits, and iterate this work-flow until finding a solution. The whole process is manual, laborious and error-prone since numerous edits should be tried.

One of the contributions of this work is an approach to annotate the LaTeX source files to include variability information. This way, variation points can determine, for instance, the inclusion or exclusion of text, different alternatives to present the same content, variable numerical values to specify the space between elements of the document, or changing the size of a figure. A variability model can then pilot the generation of final documents (*i.e.,* PDF outputs) by setting configurations.

A problem remains, however. Out of the possible paper variants, writers do not know in advance which variants will meet constraints (*e.g.,* page limits). We could generate all possible paper variants, but in practice, this case is unlikely given the combinatorial explosion of possible configurations and the amount of time or resources we have at our disposal. As a result, the exploration of possible paper variants is usually limited to a small sample. Moreover, writers can hardly predict in advance the precise effects of some modifications. In particular, we want to avoid (combinations of) edits that will always lead to a page exceed. Apart from the variability annotations, the second main contribution of this work is an approach to *learn* constraints from a random sample of configurations and retrofit the knowledge acquired directly into the variability model. This way, the space of possible configurations is restricted by construction to a good approximation. In particular, we can preclude the set of modifications leading to overrun page limits. Users can then configure their papers (*e.g.,* for controlling the aesthetics) or pick a (random) paper variant that meets constraints *e.g.,* page limits.

As a summary, the contributions of this work are:

- An approach to manually annotate the LaTeX source with *variability* information to automate the derivation of document variants.
- An automatic process using machine learning to learn how to meet the constraints using the annotated LaTeX source.
- We report on two real usages of VaryLaTeX for a scientific paper and for a curriculum vitae with their corresponding constraints.
- We make VaryLaTeX source code and experiments' data publicly available.
  https://github.com/FAMILIAR-project/varylatex

[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.*
[2] A PhD comic nicely illustrates the situation: http://phdcomics.com/comics.php?f=1971

This paper is structured as follows: Section 2 presents VaryLaTeX 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 ACK is set to true (*i.e.,* we want to include the acknowledgments text that we made optional) and BOLD_ACK is also set to true, then, the corresponding LaTeX will be:

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

Numerical values can also be used. If 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]...%
```

Users can vary various elements of a LaTeX source file: a footnote, the setting of a package for formatting code snippets, the size of a figure, *etc.* We give some examples in Figure 2a and Figure 2b. 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, bref_size can take real values between 0.7 and 1.0 while vspace_bib can take real values between 1.0 and 5.0. Specific constraints can also be added, as the logical implication between BOLD_ACK and ACK (in the example, BOLD_ACK is a child of 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 ACK and numerical values for bref_size and 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. Based on a

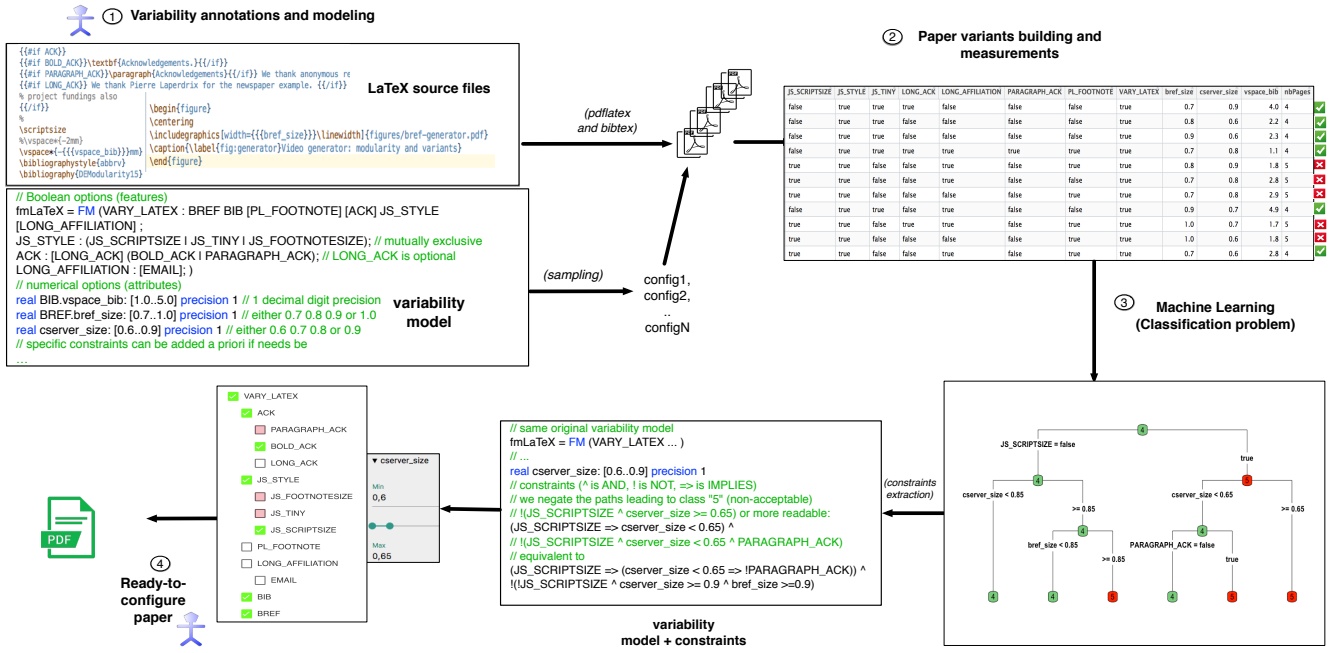
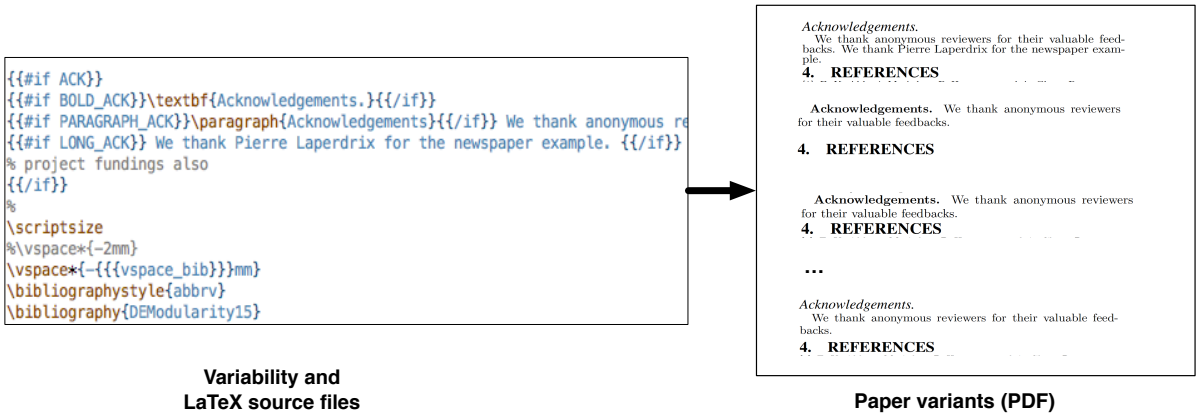


Figure 1: Varying LaTeX: What are the correct combinations of options leading to paper variants that meet page limits?



(a) Variability annotations and excerpt of some possible paper variants

```
\lstdefinelanguage{JavaScript}{
  keywords={typeof, new, true, false, catch, function, return, null, catch, switch, var, if, in, while, do, else, case, break},
  keywordstyle=\color{blue}\bfseries,
  basicstyle=\ttfamily{{#if JS_SCRIPTSIZE}}\scriptsize{{/if}}{{#if JS_TINY}}\tiny{{/if}}{{#if JS_FOOTNOTESIZE}}\footnotesize{{/if}},
```

```
{{#if PL_FOOTNOTE}}\footnote{We are considering "product lines" in a broad sense,
```

```
\begin{figure}
\centering
\includegraphics[width={{{bref_size}}}\linewidth]{figures/bref-generator.pdf}
\caption{\label{fig:generator}Video generator: modularity and variants}
\end{figure}
```

(b) Users can vary the font size of a code snippet, activate a footnote, vary the font size of a figure, *etc.*

Figure 2: Variability annotations within LaTeX source files

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 bref_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

VaryLaTeX 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[3]. In this section, we go through the different libraries and languages used to develop VaryLaTeX.

[3] https://github.com/FAMILIAR-project/varylatex

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 TeX 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 Trimou[5] a templating engine implementation written in Java. Trimou 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.→ 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.→ 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 VARYLATEX

We used VaryLaTeX in informal contexts for testing our implementation, prototyping and exploring the potential of the idea. Also, we applied VaryLaTeX 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] https://mustache.github.io/
[5] http://trimou.org
[6] http://www.minizinc.org/
[7] http://www.choco-solver.org/

up, a manual constraint inference based on manual tries and errors had strong limitations. Therefore, we decided to use VaryLaTeX 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 values 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 style of programs' listing (three possible values: scriptsize, tiny, footnotesize) and for changing the formatting of acknowledgments. Other Boolean options were used for keeping or deactivating some text. It should be noted that we did not use constraints between Boolean or numerical options, but it is something our approach supports. Users simply have to specify constraints in the variability model. For example, users can state that a specific sentence cannot be activated unless another paragraph is activated elsewhere.

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[8], 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. VaryLaTeX saved us time and resources. Second, VaryLaTeX 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 optionals. These options aimed to keep or remove some bibliography references and some other information about the career. In total, we only had 32 configurations. We used VaryLaTeX 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 [11]. 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 [7], pure::variants [4] and Gears [9] 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 [9] and a bridge in Gears [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 VaryLaTeX.

There are numerous works for predicting the performance of configurations (*e.g.,* [5, 12, 13, 17, 19]). We share the need to learn

[8] Our observations and results are on average, since we repeated the experiments 10 times.

[9] pure::variants: https://www.pure-systems.com/products/extensions/pure-variants-connector-for-microsoft-office-311.html

[10] Gears: http://www.biglever.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?" VaryLaTeX builds upon our previous works [14–16] and shows a different applicability of variability and machine learning techniques.

6 CONCLUSION

We presented VaryLaTeX, 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 VaryLaTeX.

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^2 project: FUI the Île-de-France region and BPI in France.

REFERENCES

[1] Mathieu Acher, Guillaume Bécan, Benoit Combemale, Benoit Baudry, and Jean-Marc Jézéquel. 2015. Product lines can jeopardize their trade secrets. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'15)*. 930–933.

[2] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. 2013. FAMILIAR: A Domain-Specific Language for Large Scale Management of Feature Models. *Science of Computer Programming (SCP)* 78, 6 (2013), 657–681.

[3] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: a literature review. *Information Systems* 35, 6 (2010), 615–708.

[4] Danilo Beuche. 2010. Modeling and Building Software Product Lines with pure::variants. In *SPLC Workshops*. 296.

[5] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wasowski. 2013. Variability-aware performance prediction: A statistical learning approach. In *ASE*.

[6] Mikolás Janota, Goetz Botterweck, and João Marques-Silva. 2014. On lazy and eager interactive reconfiguration. In *The Eighth International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '14, Sophia Antipolis, France, January 22-24, 2014*. 8:1–8:8. https://doi.org/10.1145/2556624.2556644

[7] Christian Kästner, Thomas Thüm, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. 2009. FeatureIDE: A tool framework for feature-oriented software development. In *ICSE*. IEEE, 611–614.

[8] Ebrahim Khalil Abbasi, Arnaud Hubaux, Mathieu Acher, Quentin Boucher, and Patrick Heymans. 2013. The Anatomy of a Sales Configurator: An Empirical Study of 111 Cases. In *CAiSE'13*.

[9] Charles W. Krueger and Paul C. Clements. [n. d.]. Systems and software product line engineering with BigLever software gears. In *SPLC 2012: Volume 2*. https://doi.org/10.1145/2364412.2364458

[10] Wei-Yin Loh. 2011. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1, 1 (2011), 14–23.

[11] Jabier Martinez, Jean-Sebastien Sottet, Alfonso Garcia Frey, Tewfik Ziadi, Tegawende Bissyande, Jean Vanderdonckt, Jacques Klein, and Yves Le Traon. 2017. Variability Management and Assessment for User Interface Design. In *Human Centered Software Product Lines*, Alfonso Garcia Frey Jean-Sebastien Sottet and Jean Vanderdonckt (Eds.). Springer International Publishing, 81–106.

[12] A. Sarkar, Jianmei Guo, N. Siegmund, S. Apel, and K. Czarnecki. 2015. Cost-Efficient Sampling for Performance Prediction of Configurable Systems (T). In *ASE'15*.

[13] Norbert Siegmund, Marko RosenmüLler, Christian KäStner, Paolo G. Giarrusso, Sven Apel, and Sergiy S. Kolesnikov. 2013. Scalable Prediction of Non-functional Properties in Software Product Lines: Footprint and Memory Consumption. *Inf. Softw. Technol.* (2013).

[14] Paul Temple, Mathieu Acher, Jean-Marc Jézéquel, and Olivier Barais. 2017. Learning-Contextual Variability Models. *IEEE Software* (nov 2017). https://hal.inria.fr/hal-01659137

[15] Paul Temple, Mathieu Acher, Jean-Marc A Jézéquel, Léo A Noel-Baron, and José A Galindo. 2017. *Learning-Based Performance Specialization of Configurable Systems*. Research Report. IRISA, Inria Rennes ; University of Rennes 1. https://hal.archives-ouvertes.fr/hal-01467299

[16] Paul Temple, José A. Galindo, Mathieu Acher, and Jean-Marc Jézéquel. 2016. Using Machine Learning to Infer Constraints for Product Lines. In *Proceedings of the 20th International Systems and Software Product Line Conference (SPLC '16)*. ACM, New York, NY, USA, 209–218. https://doi.org/10.1145/2934466.2934472

[17] Pavel Valov, Jianmei Guo, and Krzysztof Czarnecki. [n. d.]. Empirical comparison of regression methods for variability-aware performance prediction. In *SPLC'15*.

[18] Yingfei Xiong, Arnaud Hubaux, Steven She, and Krzysztof Czarnecki. 2012. Generating Range Fixes for Software Configuration. In *34th International Conference on Software Engineering*.

[19] Yi Zhang, Jianmei Guo, Eric Blais, and Krzysztof Czarnecki. [n. d.]. Performance Prediction of Configurable Software Systems by Fourier Learning (T). In *ASE'15*.