Janiform Intra-Document Analytics for Reproducible Research

Jens Dittrich Patrick Bender

Saarland University
infosys.cs.uni-saarland.de

ABSTRACT

Peer-reviewed publication of research papers is a corner stone of science. However, one of the many issues of our publication culture is that our publications only publish a snapshot of the final result of a long project. This means, we put well-polished graphs describing (some) of our experimental results into our publications. However, the algorithms, input datasets, benchmarks, raw result datasets, as well as scripts that were used to produce the graphs in the first place are rarely published and typically not available to other researchers. Often they are only available when personally asking the authors. In many cases, however, they are not available at all. This means from a long workflow that led to producing a graph for a research paper, we only publish the final result rather than the entire workflow. This is unfortunate and has been lamented upon in various scientific communitites. In this demo we argue that one part of the problem is our dated view on what a "document" and hence "a publication" *is*, *should*, and *can be*. As a remedy, we introduce portable database files (PDbf). These files are janiform, i.e. they are at the same time a standard static pdf as well as a highly dynamic (offline) html-document. PDbFs allow you to access the raw data behind a file, perform portable OLAP-style analysis, and reproduce your own graphs from the raw data — all of this *within* a portable document. We demo a tool allowing you to create PDbfs smoothly from within LaTeX. This tool allows you to preserve the connection of raw data to its final graphical output through all stages of the workflow. Notice that this pdf already showcases our technology: rename this file to ".html" and see what happens (currently Firefox, Chrome, or Safari on a Desktop machine).

1. INTRODUCTION

Irreproducibility is a problem frequently lamented upon in various scientific communites [**?**, **?**]. In the context of computer science it has recently been coined "The Real Software Crisis" [**?**]. The database community has identified it more than ten years ago and is attacking it through repeatability committees, e.g. [**?**]. In these committees a separate committee reruns the experiments of accepted papers using the datasets and code provided by the authors. Obviously, given the sheer size and complexity of some projects, in many cases these boils down to blackbox testing, i.e. it can neither be tested if the tested code actually implements the algorithms presented in the paper nor whether the code reports results in a proper way. Another problem of repeatability committees is that they cannot remedy inherent publication bias: "reviewers don't like negative results". Hence, for an experimental evaluation you need the "right" queries, the "right" dataset and the "right" baselines. This naturally leads to a flood of papers with positive results. And to papers where the "improvements don't add up" [**?**]. Publication bias was attacked by the inauguration of Experiments&Analysis papers at (P)VLDB. These kind of papers reevaluate existing work in a uniform setting and may also publish negative results. These experimental evaluations may then serve as landmarks in the flood of papers with (overly) positive results giving clear advice on the strength and weaknesses of a particular method.

This small demo is neither the place to even summarize nor defend the different arguments in the debate on repeatability and our experimental culture. It is an emotional topic where the esteemed reader of these lines probably has strong opionions in one way or the other. This is just fine. In the following, we will simply accept that there is a problem [**?**, **?**][1]. And that this problem is calling "for a new model for the way how we publish our results" [**?**]. Handling this problem can be regarded an instance of "small data" [**?**, **?**][2].

2. SIGNIFICANCE OF THE CONTRIBUTION

We believe that our contribution is significant for the following reasons:

1. **Portable DataBase Files.** We provide Portable DataBase Files, a general model to overlay a static pdf document with additional highly dynamic content. In order to specify an overlay, we simply require access to a static document S, dynamic content D, and a PDbF-configuration file C defining where to place the dynamic content. An example of D could be a relational database file and an appropriate visualization of some of its content, e.g. a bar chart visualizing measurements collected in a table.

2. **PDbF-Compiler.** We provide a compiler taking as its input the triple (S, D, C). Our compiler outputs a janiform document. That document is **at the same time** a valid pdf **and** a valid html-document. Thus, if you open the file with a pdf-viewer, you will see the static content, i.e. only the S-part.

[1] Just read the "ten simple rules for reproducable results" [**?**] and then ask yourself how little of this we follow for our papers.

[2] Also notice an upcoming Dagstuhl perspectives workshop on Artifact Evaluation for Publications [**?**] where the first author participates.

However, if you rename the file to ".html" and open it with a Web browser, you will be able to inspect the dynamic part, i.e. D and S.

3. **Full LaTeX integration.** We instrument LaTeX to output not only the static pdf-file S, but also a valid PDbF-configuration file C. In addition, we provide an extension to LaTeX allowing you to create graphs directly from within LaTeX — without requiring additional tools. Like that we are able to preserve the connectivity of raw data and graphs *through the LaTeX compiler*. In addition, this process creates dynamic variants of the graphs as a side-effect, i.e. the D-part. This means, the user can create her graphs seamlessly without worrying about different representations of graphs in static and dynamic representations. The result of this instrumentation is again a triple (S, D, C) which can be fed into our PDbF-Compiler (see Contribution 2) to create a janiform PDbF-document.

4. **Preservation of Raw Data and Graph-Connectivity.** Our compilers preserve the connection between raw data and the graphs and/or tables produced from that raw data. In addition, we are able to ship that part of the workflow, i.e. data, graphs, and the code producing the graphs within a single "document".

5. **Alternative Dynamic Views on the Data.** Our technology allows you to perform **offline** OLAP-style analytics on the **data shipped within the document**. All you need is a Web Browser (currently Firefox, Chrome, or Safari on a desktop machine). We support a rich feature list. See Section 4 for our currently supported features (as of March 31, 2015).

6. **Longterm Preservation of Raw Data.** As our tools embed the raw data within the publication, PDbF-documents naturally archive the raw data with the document. Therefore, the raw data may be "downloaded" directly from within the PDbF-document.

7. **Impact On Research in General.** We believe that our technology may not only be interesting to the database community. Our tool may be interesting for all research communities working with experimental data. Therefore, we are planning to open source our tool upon publication of this demo paper.

3. THE PDBF FRAMEWORK

3.1 Janiform File Format

How is it possible to create a single file that may both be interpreted as a valid pdf document and a valid html document? The core idea is to create a document where complementary parts of the file are ignored by the different applications.The core structure of a PDbF is shown in Figure 1.

A **PDF reader** has the following **perspective** on this file: It reads the *magic number*s "%PDF-1.5%" (line 1). Then it locates the Xref table at end of the file. That Xref contains a dictionary of all objects in this PDF file except for the dummy object (lines 3 to 8) which contains the HTML part. As the dummy object is not referenced by another object, it is never read by PDF viewers. Hence, the file is displayed as a valid PDF.

An **HTML browser** has the following **perspective** on this file: It reads lines 1 and 2. The text "%PDF-1.5%" is displayed at this point. Lines 2–5 are ignored as they are an HTML comment. The same happens for lines 6 and all following lines until the end of the document. The HTML comment is acually never closed, however, browsers are not very strict with such things. The same happens for line 1, because normally there should not be any content before the HTML tag. Hence, the HTML content is displayed.

```
1 %PDF-1.5
2 %<html><!--
3 1337 0 obj
4 stream
5 -->
  HTML PAGE
6 <!--
7 endstream
8 endobj
  PDF DOCUMENT
```

(a) PDF perspective: the pdf reader interprets the green part, everything else is treated as a comment

```
1 %PDF-1.5
2 %<html><!--
3 1337 0 obj
4 stream
5 -->
  HTML PAGE
6 <!--
7 endstream
8 endobj
  PDF DOCUMENT
```

(b) HTML perspective: the HTML browser interprets the green part, everything else is treated as a comment

Figure 1: Structure of the janiform PDbF format

3.2 Compiler Architecture

A flow chart describing how our different processing steps are invoked and how the different compilers interact is shown in Figure 2. The figure shows the entire workfow to compile a tex-file into a PDbf.

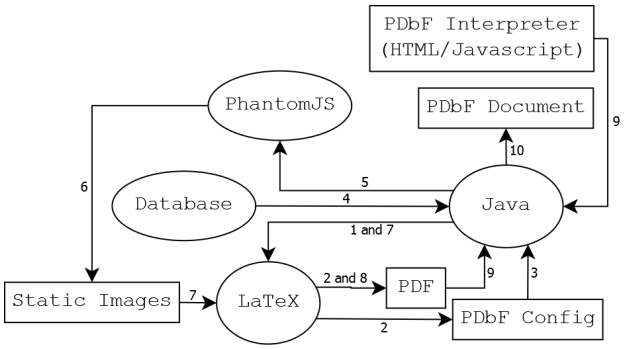

Figure 2: Compilation steps of the PDbF compiler framework

(1.) Java invokes the LaTeX Compiler.
(2.) The LaTeX compiler outputs the PDbF configuration file and a draft version of the PDF document.
(3.) Java reads the PDbF configuration file and does some postprocessing.
(4.) Java reads all tables from the database. The database is specified as a source in the PDbF configuration file.
(5.) Java invokes PhantomJS to create static snapshots of all graphs automatically.
(6.) PhantomJS outputs a static image for every dynamic object present in the PDbF configuration file.
(7.) Java invokes the LaTeX compiler a second time to obtain the final placements of all static snapshots.
(8.) The LaTeX compiler outputs the final PDF document with images from Step 6.
(9.) Java reads the final PDF document as well as the PDbF interpreter ???.

(10.) Java reads the final PDF document, the finalized PDbF configuration, and the PDbF interpreter and outputs the resulting PDbF.

cite dygraph http://dygraphs.com/, alasql https://github.com/agershun/alasql, pivot tables https://github.com/nicolaskruchten/pivottable, codemirror https://codemirror.net/, jQuery https://jquery.com/, PDF.js https://mozilla.github.io/pdf.js/

security aspects?

performance aspects, even an intuition would help, table with compile time?
space:
2.7MB + 2x PDF size time:
TODO

4. PDBF FEATURES

(currently supported)

4.1 support for DESKTOP browsers and pdf viewers

We have tested our PDbF files with the following browsers: Chrome 41, Opera 28, Firefox 36 and Safari 8.

4.2 pivot tables

4.3 bar plots

4.4 Raw Data Access

Get raw data, source code
raw data can be exported as csv with header
maybe other file attachements?

JD: bitte latex code-Beispiel zur Erzeugung eines Graphen zeigen

```
\lineChart[width=\textwidth, height=0.8\textwidth,
xunit=Date, yunit={Runtime [in sec]}]{SELECT date,
runtimeA AS engine_A, runtimeB AS engine_B FROM data2;}
```

5. THE DEMO

Major part of the demo already in your hands, the other shown at VLDB.

5.1 The PDbF File Format

This is Contributions 1, 4, 5, and 6. This part of the demo is already contained in this document. We invite the reader and (also the audience at VLDB) to change the suffix of this file from ".pdf" to ".html" (on your desktop computer). Your desktop browser will open and you will be able to see the dynamic features explained in Section 4. Notice that the dynamic content is completely offline and runs entirely in your Web browser's sandbox (Firefox, Chrome or Safari).

5.2 The PDbF Compiler

This is Contribution 2, see also Section **??**. We will invite the audience to create PDbF-files themselves on arbitrary input documents. These documents can then be enriched with dynamic content using our compiler. The final result may be viewed with a desktop Web browser.

5.3 The LaTeX Compiler

This is Contribution 3, see also Section **??**. We invite readers to bring their own LaTeX files. We will show them how to prepare their tex-documents in order to be able to define graphs directly on the raw data. We will demonstrate the seamless integration of our technology into existing LaTeX compilers.

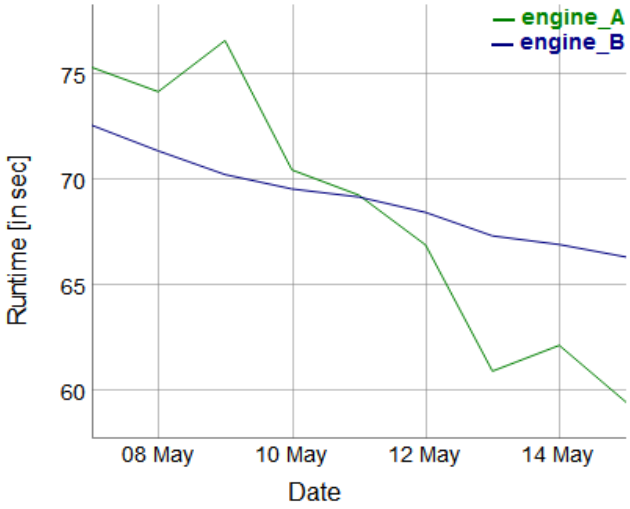

Figure 3: Multi-column Line Chart

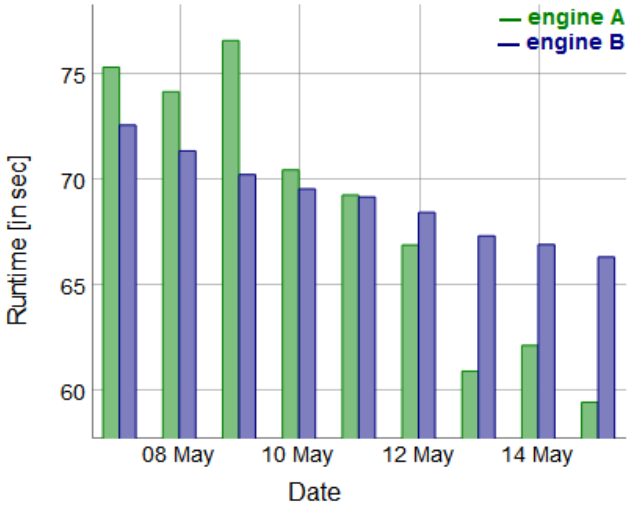

Figure 4: Multi-column Bar Chart

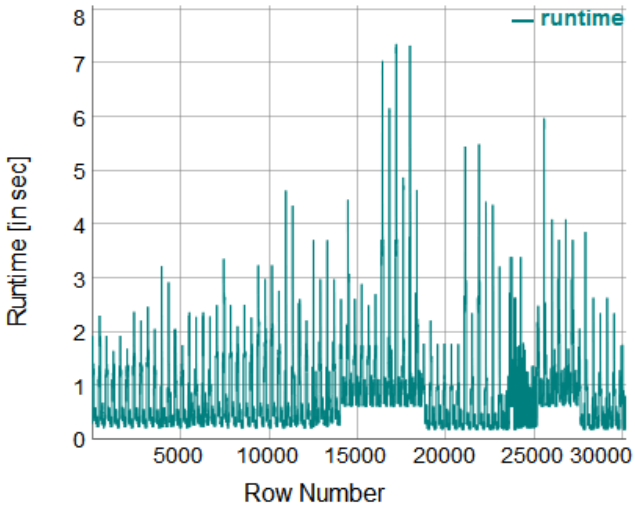

Figure 5: Lot of points

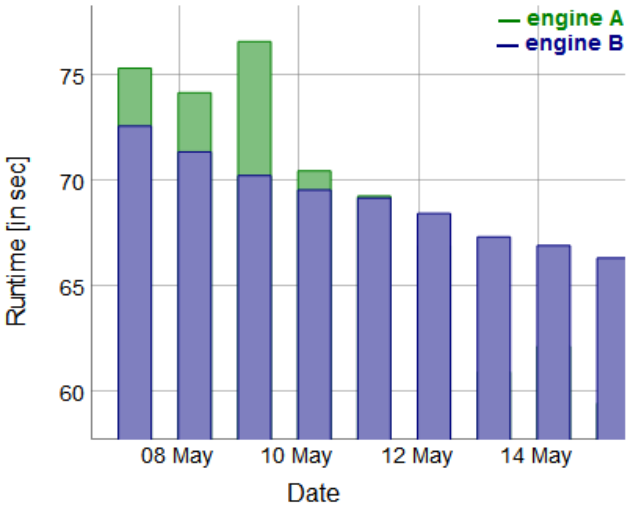

Figure 6: Multi-column overlapping Bar Chart

Figure 7: Pivot table

6. RELATED WORK

hmm, how much should we really say here?

SIGMOD reproducability initiatives and papers (two, right?)

upcoming Dagstuhl perspectives workshop

experiments and analysis track and its problems: access to data, access to code

Pweave http://mpastell.com/pweave/index.html

7. CONCLUSIONS

This demo opened the book for portable database files (PDbF). PDbFs allow you to embed raw data into a document while preserving the entire data-pipeline from raw data to graphs. This allows readers to perform in-document OLAP-style analytics on the published document. PDbFs are janiform documents that are at the same time a valid pdf and a valid html document. Thus they can be viewed in either way. This pdf is an example of such a janiform PDbF. In additon, the conference demo show-cases two compilers allowing you to seamlessly create PDbFs, also from within LaTeX.

As part of future work we want to research how to include other parts of the research pipeline as part of the pdf. For instance, in future, one might consider to embed a virtual machine emulator, an operating system image, and all software that is required to run the original code *within the pdf* (~3GB of data). This may sound infeasible in 2015. But 4K video streaming over the Internet sounded equally silly in 1995.

8. REFERENCES