Bachelorthesis
in Media Computer Science

Modern Architecture and Improved UI for Tables of BenchExec

Laura Sofie Bschor

Aufgabensteller:	Prof. Dr. Dirk Beyer
Betreuer:	Dr. Philipp Wendler
Abgabedatum:	02.10.2019

Erklärung

Hiermit versichere ich, dass ich diese Bachelorthesis selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, den 02.10.2019

.....
Laura Sofie Bschor

Abstract

Benchmarking is a common practice in sciences. Benchmarking results are typically big data sets with hundreds and thousands of values. For a user of the benchmarking tool to extract and find the results they are interested in the results must be displayed in a clear way. This thesis is concerned with providing an interactive and clear presentation of results of the tool BenchExec. As there is already an implementation for this - which is outdated in design and architecture and without a holistic concept - this is taken as an idea provider for creating a new state-of-the-art frontend application for presenting and interacting with the results of BenchExec. Its requirements and the decisions about development were generated by the users themselves, related work, a theoretical background and technical researches and comparisons. The new version was implemented in parallel with the completion of this thesis. Finally the new version is compared to the original to see if there are improvements or regressions and to provide a solid basis for continuous human-centered development in the future.

Contents

Chapter 1

Introduction

A common practice for researchers, developers and tool competitors is benchmarking. Benchmarking is used to evaluate and compare algorithms, tools and features as well as different settings, configurations or inputs of a given tool. Hundreds and thousands of runs can be required for an evaluation in a single comparison. Results of these benchmarking methods have to be exact and clearly presented. This can be difficult because the results can be a big set of data with varying sizes, values or settings. BenchExec, a project of the Software Systems Lab of the Ludwig-Maximilians-University in Munich is an open-source implementation of a benchmarking framework. It is tool-independent and ready to use. So you are able to benchmark and measure tools and resources. One of the three major features is the table-generator, which generates interactive tables and plots to visualize the BenchExec's XML output results of one or more executions as a HTML or CSV file [2, 8]. BenchExec table-generator is written in Python and generates a frontend application wherein the results are displayed and can be interacted with. For generating an HTML template is used and has grown with time and incoming feature requests, made by the users of BenchExec. The basic implementation of the table including markup, style-sheet and JavaScript in one file was done in 2015. As a result it is now a collection of features and functions without a holistic concept for handling data, states, user needs and the architecture of the code.

The goal of this thesis is to provide a state-of-the-art, holistic and intuitively usable application which visualizes the results of BenchExec clearly and lets the user interact with them to get the needed information fast and easy. To achieve this, we derive requirements based on analysis about related work and the tool environment, the user issues and researches concepts surrounding modern frontend architectures and frameworks. Decisions about the implementation were made based on the requirements. The result of these deci-

sions is the new version of BenchExec HTML tables, which is implemented in parallel with the completion of this thesis. In the end the fulfillment of requirements will be evaluated through the user's satisfaction.

Chapter 2

Motivation

Interaction and dynamic elements within the HTML files are directly linked to JavaScript - a client-side programming language (so called script language). It is directly interpreted by the browser, but even so it works without any internet connection and was developed in 1995 by the company NETSCAPE [3].

Since then, many things have changed. For example different frameworks have been invented, JSON (JavaScript Object Notation) has taken a big part in the daily business of JavaScript developers and the language itself has become one of the most popular programming languages. It has become hard to keep up-to-date with approaches, technologies and framework versions. But they are, as well as the words *usability*, *user-experience* or *human-centered design*, no longer indispensable from the world of web development. In the following chapter, we will work out why a new version of BENCHEXEC HTML tables should be implemented and what the requirements for this new version are. To gain knowledge about these requirements the first part introduces approaches about focusing on the user during the process of development. After that related work is analyzed to get ideas of similar projects and visualizations. Afterwards the technical status and the environment of TABLE-GENERATOR and its template is analyzed to see what has to be improved. At last results of a survey are analyzed to gain knowledge about the users issues, needs and workarounds. This survey was sent around to the users of BENCHEXEC before implementing a new version of BENCHEXEC HTML tables. To get to know the users issues was only one goal of the survey: The users were also asked about their opinions of the old version. After the implementation another survey was sent around about the new version. With the results of both surveys user's opinions of the two versions of BENCHEXEC HTML tables can be compared to evaluate if the implementation was an improvement or a regression for the user. The details of this comparison are described in Chapter 5.

2.1 User in the Focus of Development - Theoretical Background

At first, the main development approaches and associated terms have to be declared to get a common conceptual basis.

2.1.1 User Experience

By interacting with any kind of tool, system or product (in the following called *system*), users have perceptions, reactions and emotions in relation to it [4]. They can prejudge or be excited before using, they can like or hate it, can have physiological or psychological reactions while using it and they will have an opinion and a feeling after using it. *User experiences* (short: UX) are the sum of all reactions and emotions in relation to the used. They are the consequence of a combination of users (with their previous knowledge, their preferences, behaviors, current states and the context of use) and the system (with its brand image, the functionalities and the general appearance). First and foremost, user experience cannot be objectively judged, but a developed system can aim to provide a good user experience. This includes considering the user in your decisions about design, shaping, behavior and functionalities when developing the system. Good user experience provides a map for the users of where they are in the system and how they always can get back to the starting point [1]. This refers to a mental model which is generated by the users themselves: They interact with and experiences the system, its behavior and reactions, they read and observe everything that comes with the system and form their opinion about what it does, what it is and how it can be used. If this mental model is not good and something goes wrong, the users will not understand why it did not go the way they wanted and can only try. But otherwise, if the system has a logical, clear and consistent concept communication between system and users will be easy and clear.

2.1.2 Usability

The nature of responses and emotions associated with the users experience just described may be expressed as *usability* as far as it is seen with the goals the users tries to achieve with the system [4]. Good usability does not only mean the quality of a system or if it is easy to use [5, 6]. It includes the whole user experience, also with the satisfaction when working with the system. Usability is the rating of users experience. It depends on the simplicity with which user can achieve a goal through the system. If this happens effectively with efficiency and satisfaction, the user experience will be good, so the usability

will be high. Benyon defines usability as the "quality of the interaction in terms of parameters such as time taken to perform tasks, numbers of errors made and the time to become a competent user" [1] and to be more concrete, Jakob Nielson writes about five quality components[1]:

- Learnability: If it is the first time users enter the system: How easy is it to accomplish basic tasks?
- Efficiency: How quickly can users achieve goals once they have gotten to know the system?
- Memorability: How easily can users re-establish proficiency if they do not use the system regularly and did not have entered it for a time?
- Errors: How many errors are made by the users and how serious they are? How bad did the users take it?
- Satisfaction: How well do the users feel after using the system?

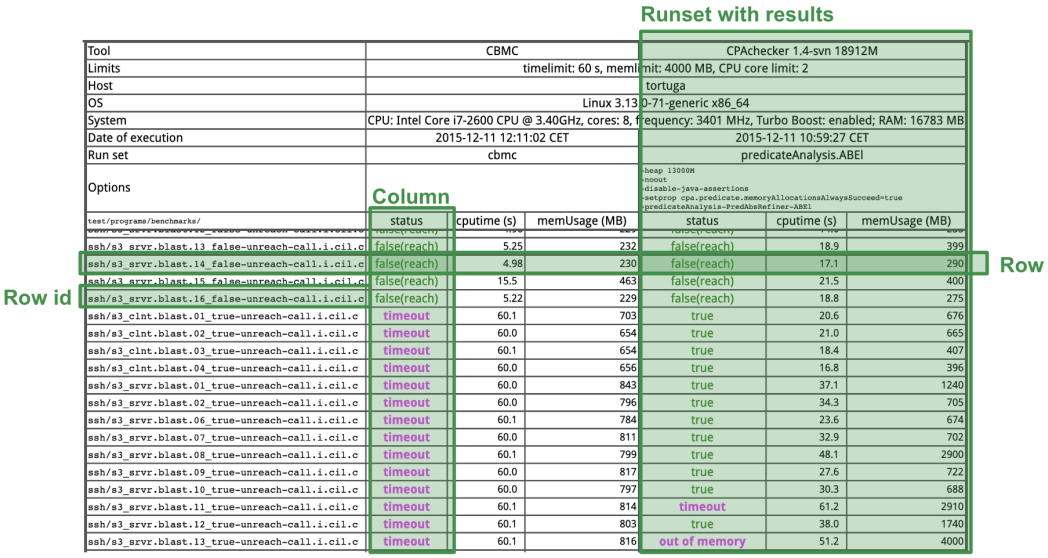
To write it as a mathematical function usability is the sum of effectiveness (*"is the user's goal achieved?"*), efficiency (*"how much time did it take to complete the task, how many errors happened, how high was the amount of effort"*) and satisfaction (*"how high was the level of comfort users feel when using the system"*) [4].

2.1.3 Human-centered Design

Human-centered design is the main approach which we are following to work out how the new version of BenchExec HTML tables should work and look like. It is an approach (standardized and recorded in DIN EN ISO 9241-210 [4]) for developing interactive systems in an usable and appropriate way. As the title tells us, users with their needs and requirements are in the focus of the whole process and every step. This focus in combination with techniques of ergonomics for usability shall rise the effectiveness and efficiency of users and improve their well-being and satisfaction while working with the system. To achieve these goals the following principles will be adhered to [4]:

- Understand your users, their tasks and working environment: Who is using my product, what is the target group, what are their needs and goals. Gain and include this knowledge into the process and your development.

[1] `https://www.nngroup.com/articles/usability-101-introduction-to-usability/`



Tool	CBMC			CPAchecker 1.4-svn 18912M		
Limits	timelimit: 60 s, memlimit: 4000 MB, CPU core limit: 2					
Host	tortuga					
OS	Linux 3.13.0-71-generic x86_64					
System	CPU: Intel Core i7-2600 CPU @ 3.40GHz, cores: 8, frequency: 3401 MHz, Turbo Boost: enabled; RAM: 16783 MB					
Date of execution	2015-12-11 12:11:02 CET			2015-12-11 10:59:27 CET		
Run set	cbmc			predicateAnalysis.ABEl		
Options				-heap 13000M -noout -disable-java-assertions -setprop cpa.predicate.memoryAllocationsAlwaysSucceed=true -predicateAnalysis-PredAbsRefiner-ABEl		
test/programs/benchmarks/	status	cputime (s)	memUsage (MB)	status	cputime (s)	memUsage (MB)
ssh/s3_srvr.blast.13_false-unreach-call.i.cil.c	false(reach)	5.25	232	false(reach)	18.9	399
ssh/s3_srvr.blast.14_false-unreach-call.i.cil.c	false(reach)	4.98	230	false(reach)	17.1	290
ssh/s3_srvr.blast.15_false-unreach-call.i.cil.c	false(reach)	15.5	463	false(reach)	21.5	400
ssh/s3_srvr.blast.16_false-unreach-call.i.cil.c	false(reach)	5.22	229	false(reach)	18.8	275
ssh/s3_clnt.blast.01_true-unreach-call.i.cil.c	timeout	60.1	703	true	20.6	676
ssh/s3_clnt.blast.02_true-unreach-call.i.cil.c	timeout	60.0	654	true	21.0	665
ssh/s3_clnt.blast.03_true-unreach-call.i.cil.c	timeout	60.1	654	true	18.4	407
ssh/s3_clnt.blast.04_true-unreach-call.i.cil.c	timeout	60.0	656	true	16.8	396
ssh/s3_srvr.blast.01_true-unreach-call.i.cil.c	timeout	60.0	843	true	37.1	1240
ssh/s3_srvr.blast.02_true-unreach-call.i.cil.c	timeout	60.0	796	true	34.3	705
ssh/s3_srvr.blast.06_true-unreach-call.i.cil.c	timeout	60.1	784	true	23.6	674
ssh/s3_srvr.blast.07_true-unreach-call.i.cil.c	timeout	60.0	811	true	32.9	702
ssh/s3_srvr.blast.08_true-unreach-call.i.cil.c	timeout	60.1	799	true	48.1	2900
ssh/s3_srvr.blast.09_true-unreach-call.i.cil.c	timeout	60.0	817	true	27.6	722
ssh/s3_srvr.blast.10_true-unreach-call.i.cil.c	timeout	60.0	797	true	30.3	688
ssh/s3_srvr.blast.11_true-unreach-call.i.cil.c	timeout	60.1	814	timeout	61.2	2910
ssh/s3_srvr.blast.12_true-unreach-call.i.cil.c	timeout	60.1	803	true	38.0	1740
ssh/s3_srvr.blast.13_true-unreach-call.i.cil.c	timeout	60.1	816	out of memory	51.2	4000

Figure 2.1: Example Table with Names of Table Parts - Screenshot taken from `https://sosy-lab.github.io/benchexec/example-table/svcomp-simple-cbmc-cpachecker.table.html`

- Integrate the users during the whole process of development. They can give you important knowledge for acting with your future product, it's tasks, usage and context of use. Ask them directly and include their answers into your development.
- Adapt and refine your current solution with this gained knowledge. You can test your prototype with the users. Ask them about decisions to be made. Test your product in the real world with real users and get feedback to continuously improve your product.
- This process is iterative.
- During the whole development, consider the user experience.
- Build the team with people of different perspectives and knowledge.

2.1.4 The BenchExec HTML Table

As BenchExec HTML Table - and with it this work - is using a specific terminology, some of these terms will be explained in the following. To provide a consistent use of notations a look on BenchExec's architecture of a table for results is helpful and shown in Figure 2.1.

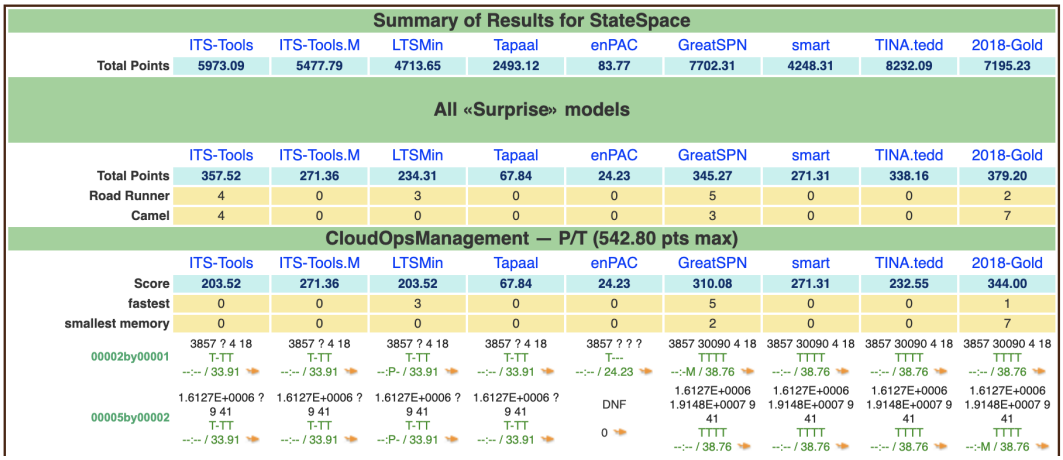
The results for one benchmarking task are displayed per *row*. Every row has an identification (*Row ID*) consisting of the filename and optional properties. A *runset* is the benchmarked tool (program) including its results by tasks. It has one or more columns which represent a parameter and its measured value for the task (e.g. cputime). In the header of the result table the properties and environment information of the runset are displayed. To generate such an interactive table `table-generator`[2] has to be called on the command line with one or more XML-files (with the results from BenchExec) handed over to compare. They can be passed through a local path or an URL. To customize the table, several flags can be added to the command (e.g. specifying the format). Table-generator links data and properties (from XML-files and command line flags) with an HTML file called `template.html` and puts out a new file in the specified format (HTML or CSV) which can be opened in a browser and used without any internet connection. This `template.html` includes DOM, styling (CSS) and behavior (implemented in JavaScript with the use of JavaScript library jQuery).

2.2 Related Work

To get an idea how benchmarking results can be visualized there are several examples discussed in the following. They are a selection of competition results from the TOOLympics at the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)[3] and the Federated Logic Conference (FLoC) Olympic Games[4]. The first example is the presentation of the computed results of Model Checking Contest 2019. The example screenshot in Figure 2.2 shows its visualization of results which starts with the summary of total points of each participatory tool. These are displayed in the columns and each row is representing the values of a model instance. Expected results are presented in a tooltip to be seen by hovering over the name of the instance. Values including rating, score and a link to the execution report are contained in a single cell. Above the table, there is an explanation to read and interpret the presented values.

The CADE ATP System Competition[5] provides results in four kinds: A compact and full summary, a detailed list of results (exemplary screenshot in Figure 2.3a) and as performance graphs (exemplary screenshot shown in Figure 2.3b) each behind a link. The tables use colors without any explanation

[2] which is one of the main features of BenchExec and written in Python
[3] `https://tacas.info/toolympics.php`
[4] `https://www.floc2018.org/floc-olympic-games/`
[5] `http://www.tptp.org/CASC/J9/`



Summary of Results for StateSpace

	ITS-Tools	ITS-Tools.M	LTSMin	Tapaal	enPAC	GreatSPN	smart	TINA.tedd	2018-Gold
Total Points	5973.09	5477.79	4713.65	2493.12	83.77	7702.31	4248.31	8232.09	7195.23

All «Surprise» models

	ITS-Tools	ITS-Tools.M	LTSMin	Tapaal	enPAC	GreatSPN	smart	TINA.tedd	2018-Gold
Total Points	357.52	271.36	234.31	67.84	24.23	345.27	271.31	338.16	379.20
Road Runner	4	0	3	0	0	5	0	0	2
Camel	4	0	0	0	0	3	0	0	7

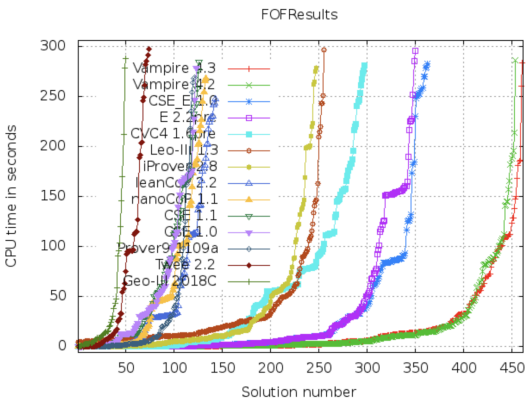
CloudOpsManagement — P/T (542.80 pts max)

	ITS-Tools	ITS-Tools.M	LTSMin	Tapaal	enPAC	GreatSPN	smart	TINA.tedd	2018-Gold
Score	203.52	271.36	203.52	67.84	24.23	310.08	271.31	232.55	344.00
fastest	0	0	3	0	0	5	0	0	1
smallest memory	0	0	0	0	0	2	0	0	7
00002by00001	3857 ? 4 18 T-TT --:-- / 33.91	3857 ? 4 18 T-TT --:-- / 33.91	3857 ? 4 18 T-TT --:P- / 33.91	3857 ? 4 18 T-TT --:-- / 33.91	3857 ? ? ? T--- --:-- / 24.23	3857 30090 4 18 TTTT --:-M / 38.76	3857 30090 4 18 TTTT --:-- / 38.76	3857 30090 4 18 TTTT --:-- / 38.76	3857 30090 4 18 TTTT --:-- / 38.76
00005by00002	1.6127E+0006 ? 9 41 T-TT --:-- / 33.91	1.6127E+0006 ? 9 41 T-TT --:-- / 33.91	1.6127E+0006 ? 9 41 T-TT --:P- / 33.91	1.6127E+0006 ? 9 41 T-TT --:-- / 33.91	DNF 0	1.6127E+0006 1.9148E+0007 9 41 TTTT --:-- / 38.76	1.6127E+0006 1.9148E+0007 9 41 TTTT --:-- / 38.76	1.6127E+0006 1.9148E+0007 9 41 TTTT --:-- / 38.76	1.6127E+0006 1.9148E+0007 9 41 TTTT --:-M / 38.76

Figure 2.2: Screenshot of a Summary of Results for MCC's StateSpace taken from `https://mcc.lip6.fr/index.php?CONTENT=results/StateSpace.html&TITLE=Results%20for%20StateSpace`

Higher-order Theorems	*Satallax* 3.2	Satallax 3.3	Leo-III 1.3	LEO-II 1.7.0
Solved/500	406/500	401/500	355/500	213/500
Av. CPU Time	32.81	32.20	38.54	17.92
Av. WC Time	32.81	32.13	16.95	17.96
Solutions	406 81%	401 80%	355 71%	209 41%
µEfficiency	266	262	134	290
µWCEfficiency	287	286	59	290
SOTAC	0.33	0.33	0.35	0.29
Core Usage	0.99	0.97	2.62	0.76
New Solved	0/0	0/0	0/0	0/0

(a) presented as summary, taken from `http://www.tptp.org/CASC/J9/WWWFiles/ResultsSummary.html`



(b) presented as plot, taken from `http://www.tptp.org/CASC/J9/WWWFiles/ResultsPlots.html`

Figure 2.3: Screenshots of Results for One CASC Competition 2018

of the colors. The detailed results for each run are displayed in the full result list. Plots visualize the CPU time of each tool for its solutions. This lets you easily recognize the rating of the competition, even so the legend overlapping the graph causes some distraction.

Some competitions present their results in more or less interactive tables like SAT Competition 2018[6] or the Reactive Synthesis Competition (syntcomp)[7]. For this they use the jQuery libraries tablesorter[8] and DataTables[9]. This allows a number of functionality, e.g. sorting of rows by values or a free text search. DataTables is also used by StarExec[10], a web-based logic solving service developed at the University of Iowa [7]. In 2019, syntcomp stored their benchmarks and results, as well as ran its competition in StarExec[11]. In addition to sorting and searching, the table of StarExec has pagination and highlighting. Besides the storing, running and presenting of competitions, it gives the possibility to add interactive plots of statistics. Examples screenshots of plots and results are shown in Figures 2.4 and 2.5.

2.3 Technical Status

The example of StarExec from Section 2.2 shows that it might be useful to include a library for offering more functionalities in the interactive result table. Tablesorter's latest version 2.17.8 was released on September 15th, 2014 which is not compatible with a modern frontend architecture. So does dataTables: The latest version (1.10.19) was released on June 22th, 2018. JQuery's latest version (3.4.1) was released May 1st, 2019 but the number of contributors and commits decreased massively. Especially in comparison to other common JavaScript frameworks like React or Angular jQuery is not any longer state-of-the-art or developing further like the comparison of github insights shows.

Figures 2.6 and 2.7 show the differences in commits per week for jQuery and React. The maximum of jQuery commits per week is nine commits which happened only once in the past year. In most of the weeks there was no or one commit per week in comparison to the weekly commits of React. There are only two weeks with one commit (in the last week of December and the first week of January). The rest of the year React has most of the times up to fifty commits per week in the week of 7th, 2019 up to 170.

Also the number of contributions to the master branch over the years since

6 http://sat2018.forsyte.tuwien.ac.at/index.php
7 http://www.syntcomp.org/
8 https://plugins.jquery.com/tablesorter/
9 https://datatables.net/
10 https://www.starexec.org/
11 http://www.syntcomp.org/

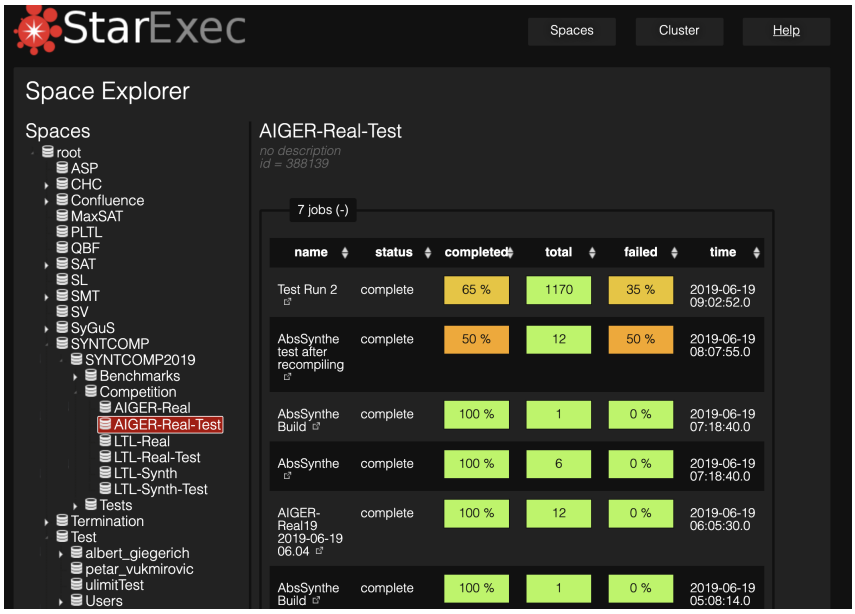


Figure 2.4: Ran Jobs of AIGER-Real-Test, a Competition of Syntcomp - Screenshot taken from `https://www.starexec.org/starexec/secure/explore/spaces.jsp?id=388139`

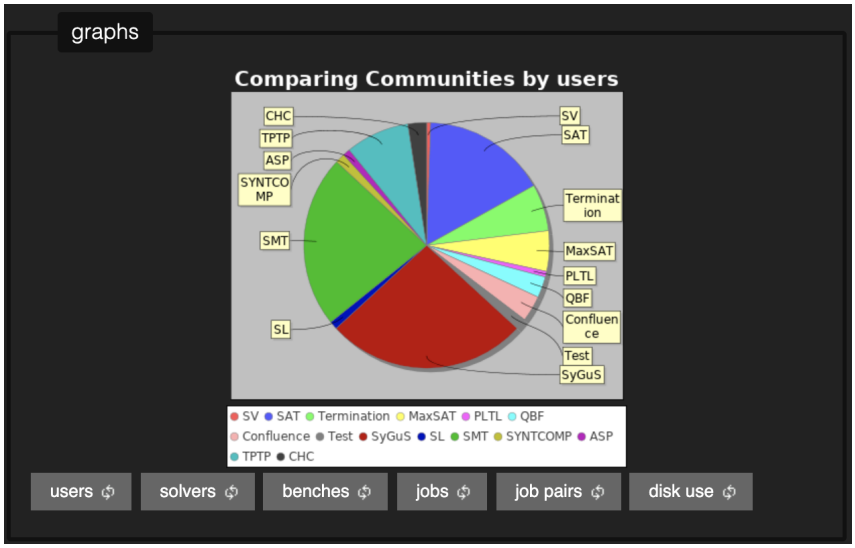


Figure 2.5: An Example for a Graph in StarExec - Screenshot taken from `https://www.starexec.org/starexec/secure/explore/statistics.jsp`

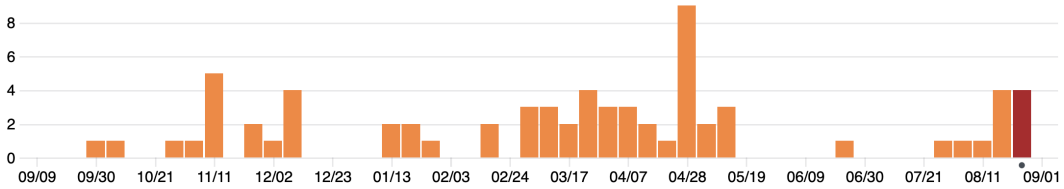


Figure 2.6: Number of Weekly Commits for jQuery in the Past Year - Screenshot taken on September 8th 2019 from `https://github.com/jquery/jquery/graphs/commit-activity`

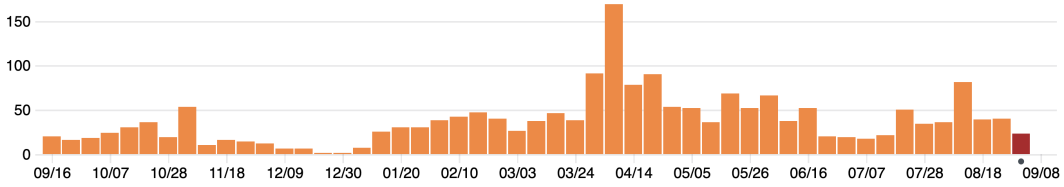


Figure 2.7: Number of Weekly Commits for React in the Past Year - Screenshot taken on September 9th, 2019 from `https://github.com/facebook/react/graphs/commit-activity`

release of jQuery and Angular are different (Figures 2.8 and 2.9): jQuery maximum number of contributions is 60, Angular's number of contributions is up to 120. September 6th 2018 GitHub.com announced the complete removal of jQuery from the GitHub.com frontend because it is not longer necessary[12]. In addition, the fact that jQuery manipulates the DOM directly can cause the problem that keeping track of all manipulations and states can become very hard as well as the fact that a big DOM might need long loading times and has

[12] `https://github.blog/2018-09-06-removing-jquery-from-github-frontend/`

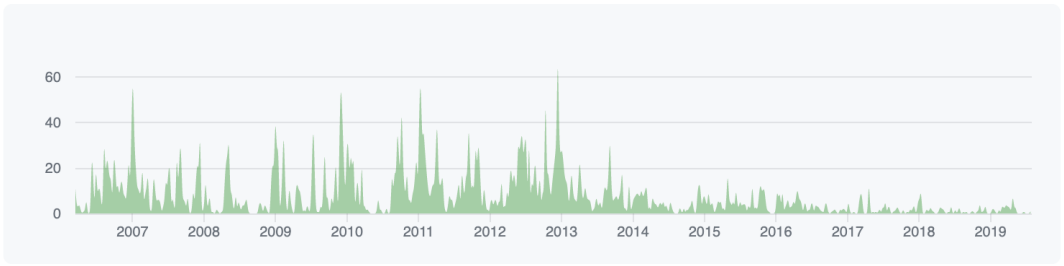


Figure 2.8: Number of jQuery Contributors from March 19th, 2006 until August 3rd, 2019 - Screenshot taken on August 4th 2019 from `https://github.com/jquery/jquery/graphs/contributors`

Figure 2.9: Number of Angular Contributors from September 14th, 2014 until August 3rd - Screenshot taken on August 4th 2019 from `https://github.com/angular/angular/graphs/contributors`

no clear structure[13]. To maintain and continuously improve the application to be state-of-the-art a structuring framework is appropriate especially because DOM, styling and functions have to be in one document at the end. While developing frontend applications the code should be well readable, understandable and clear structured whereas it needs to be bundled when providing it for the productive system. This is one major feature which comes along with modern JavaScript frameworks like React, Vue and Angular. All the beforementioned downsides of jQuery lead to rethink the whole frontend architecture and bring newer frameworks and concepts into mind.

2.4 User's Issues

As described in Section 2.1.3 we want to align our application to the needs of BenchExec's users. Therefore a first online survey (google form) was sent around for two weeks to the user of BenchExec (per mail and as a link in gitHub). 16 users participated, most of them are developers (87,5%). Besides ratings about usability, design and user-experience they were asked about their way of using BenchExec HTML tables (e.g. "When you open BenchExec HTML tables, what is the first thing you are looking at?") and had the chance to announce their requirements, needs and wishes. The answers from the survey are a relevant source of information to set the requirements. Complete results of the survey can be found in the appendix and will be picked up again when evaluating the new implementation in Chapter 5.

With regards to the usage the most relevant finding the survey should achieve is the first information the users are searching for when opening the application/table. Most of the users (56,3%) have a look at the summary first. This can take a lot of time, because the summary is on the bottom of the table

[13] `https://academind.com/learn/javascript/the-world-of-javascript/`

which can have an unlimited number of rows. Some participants of the survey also mentioned this as an answer about the most annoying feature (e.g. “first loading the whole table, with summary at the bottom”, “summary is at the bottom: need to scroll and often loading takes a while”).

Another finding about the provided features is, that most of them are used often or very often, but there are some that are hard to find. This can be interpreted as the reason of the uncontrolled growing as mentioned in Chapter 1. Because they were already desired by users and are used, it would be useless to delete this functionality but it is necessary to make them detectable and intuitively usable so that more people can benefit from them.

Loading times are in sum the most mentioned thing in the survey even without being asked. Especially in combination with the position of the summary but also with filtering and selecting (e.g. “Compiling time: it takes a lot of time to show the whole report, if there are a lot of benchmarks”, “Deselecting columns takes a long time, because the table is updated on every column I deselect, but e.g. for SV-COMP I often want to deselect all but one column”).

2.5 Requirements

Referring to the sections of this chapter some requirements can be formed which provides a guide, and later a checklist, to be fulfilled. A new application has to be created, which is intuitively usable. All given features shall be maintained to let the users be effective but the application shall help them to find all of the features and functions that they can decide whether to use them or not. To be competitive, it shall provide new features like sorting or paging. The loading time has to be reduced and the structure overthought to let the user fast and easy achieve their goal. All the previous points shall be provided through a newly implemented application based on a modern, lightweight, structuring, maintainable and fast state-of-the-art JavaScript framework.

Chapter 3

Software Architecture

The following chapter is focused on fulfilling the elaborated requirements regarding the software architecture, maintainability and performance to achieve a modern architecture of BenchExec HTML tables.

The frontend of table-generator was implemented with the use of jQuery[1] (version 1.7.1 which was released in 2011[2]). As argued in the Section 2.3 it has to be replaced by a modern lightweight, structuring and maintainable frontend framework. With this new architecture the data structure has to be adapted and handled in a different way.

3.1 Comparison of Frontend Frameworks

To replace the old one a new framework must be found. Therefore we have to clarify what a modern frontend architecture is and what shall be taken. At the moment, there are three popular options available in the world of web-development: AngularJS, React and Vue.js[3]. Some time ago frontend only meant to give a functionality and interaction to a web application. This is definitely given with jQuery. However, now users want to have reactive content and developers want to have structured code.

3.1.1 Introduction

All three frameworks solve the problems jQuery left[4]: They structure the application component based, keep track of states and render only the asked

[1] https://github.com/sosy-lab/benchexec/commits/master

[2] https://blog.jquery.com/2014/12/18/jquery-1-11-2-and-2-1-3-released-safari-fail-safe-edition/

[3] https://academind.com/learn/javascript/the-world-of-javascript/

[4] https://academind.com/learn/javascript/the-world-of-javascript/

components at runtime. This means that the framework renders only those parts of markup based on the relevant data which are needed. This works as contrasted with jQuery, which renders the complete markup and modifies it. To decide which framework to take for replacing jQuery, the afore-mentioned frameworks are introduced in the following before they are compared.

Angular (2+) Angular 2+[5] is the rewritten version of AngularJS, released in 2016 by Google. It uses TypeScript and two-way-data-binding[6]. Angular can not be dropped into any existing HTML page, has a complex project setup and is focused on creating big single-page-applications for every kind of deployment target[7]. It works with modules providing the compilation context for components, each has a purpose and responsibility. Components define views (a set of screen elements) and use services (provide the functionality)[8] [9].

React React[10] was released by Facebook in 2013 and is described as perfectly suited for creating interactive user interfaces. The logic of each component is written in JavaScript and it implements its own language for creating markup (JSX). Due to this a React component is completely written in JavaScript so the data can directly be passed, handled and changed. React components work with virtual DOMs: If the data changes, React compares virtual and existing DOM and re-renders only the necessary changes. This provides a faster performance in comparison to exchanging the whole DOM. Data can be passed through components and held in one to keep track of state and data of the whole component.

Vue (Version 2.0) Vue[11] is the only framework created by a single person not a company behind it: Evan You, an independent software developer. The first version of Vue has been published 2014, the second version - Vue(2) - was released in 2016. It is very flexible in terms of the size of application it is usable for. One can put it into existing HTML pages (progressive framework) or build complete single page applications. Vue combines the ideas of React (controlled part of the HTML to render your data/component in) and Angular

5 `https://angular.io/`

6 Two-way-data-binding describes the data is synchronized regardless of whether it is changed by the user or by the application itself (`https://angular.de/buecher/angularjs-buch/databinding/`)

7 https://academind.com/learn/javascript/the-world-of-javascript/

8 `https://academind.com/learn/angular/angular-vs-react-vs-vue-my-thoughts/`

9 `https://angular.io/guide/architecture/`

10 `https://reactjs.org/`

11 `https://vuejs.org/`

Table 3.1: Comparison of the Average Slowdown Statistics (non-keyed) of JavaScript Frameworks for One Benchmark Example

Duration for...	angular-v7.1.4 non-keyed	react-v16.6.0 non-keyed	vue v2.6.2 non-keyed
interacting with the table	1.09	1.10	1.42
startup metrics	2.27	1.20	1.00
memory allocation	1.61	1.18	1.01
summary	1.66	1.16	1.14

(separate HTML, CSS and JavaScript but place the functionality-instruction directly into the HTML code).

3.1.2 Comparison

The application should load fast and perform well when running (e.g. updating the table by filtering). It should be well structured to guarantee the maintainability and keep track of states (like the current set of displayed data etc.). With all of the three frameworks it is possible to build powerful apps with a component-based structure. Angular is designed for creating very big single page applications with good performance. React and Vue can be used as well for creating single page applications, they only do not have their focus on it. They can also be implemented in Multi-Page-Applications. To compare the performance, the framework benchmark project js-framework-benchmark [12] with statistics for the mode *non-keyed* was used. Non-keyed concerns the data-binding between data and DOM. It means, that data is replaced by new ones in elements of the DOM in comparison to *keyed*, where the elements are removed and new one would be added. For example, if we compare the function of pagination in a table: By clicking on a button to the next page, the content of all rows has to be replaced but the structure of the table stays. The keyed approach will delete all rows and add all rows as new elements in the DOM. However, the non-keyed approach leaves the DOM as it is and only replaces the data. This improves the performance and gives the reason to only compare and prefer the usage of non-keyed approaches. The slowdown describes always the duration of current framework divided by the fastest one.

In the benchmark tool we compare angular v7.1.4 with react-v16.6.0 and vue-v2.6.2 (all non-keyed) with regard to the duration of actions with a ta-

[12] `https://github.com/krausest/js-framework-benchmark`

Table 3.2: Comparison of the Popularity of React and Vue

Popularity of	Vue	React
Contributors	278	1,304
GitHub-Stars	145,663	134,099
Used by	974,844	2,311,749
Posted jobs	1356	4142

ble, start up metrics and memory allocation. The complete comparison of values can be found in Appendix A. The summary represents the average of slowdowns per framework and category of comparison. It is calculated by comparing the duration for one action (e.g. replacing all rows), and taking the fastest framework as the reference point for this action. The other frameworks are compared to this reference point. The values in Table 3.1 are the calculated average of slowdowns per category of comparison and framework. Summarizing the first section (interacting with the table like creating, swapping, updating,... rows), vue.js has the worst value with 1.42. This means the average of the duration of acting with the table in this benchmarking example in comparison to the fastest one per row. As this distance to 1.0 is quite low (what would mean, that all three are same), they are close together. Mainly the duration of selecting rows impairs the rating, to find in Appendix A.

The second section compares the startup metrics of the three frameworks like the script bootup time or the thread work cost. This output is very important because the users of BenchExec HTML tables complained a lot about the loading time of huge result tables. Here, vue.js has the best value in every task, React is a little bit worse with 1.20 but angular has an average of 2.27. This is mostly affected by the script bootup time which includes the parsing, compiling and evaluation of all scripts. In addition the weight of the application (post-compression) is higher than in React and vue.js (1.70) what is crucial because e.g. some users of BenchExec send the HTML file of the table via email.

Also the summarized memory allocation of an angular-app was higher than in React and Vue: While React and Vue are similar (1.01 and 1.02) Angular has a slowdown geometric mean of 1.61. This is higher than the others mainly influenced by the memory usage after page load with a duration of 2.46 (can be seen in Appendix A).

As React and Vue are similar in performance and usability, the decision is made by the popularity of the framework; by how easy it would be to find someone to maintain and improve the application in the future. Both frameworks were compared by their contributors, GitHub-Stars, number of

users[13] and posted jobs on Xing.com[14]. As shown in Figure 3.2, React has more contributors and posted jobs. It is also a lot more used. Only the number of GitHub-Stars is less than in Vue. Interpreting this table, the chance to find another person who is familiar with React seems to be higher than with Vue. In addition, React is developed and advanced by Facebook and not a private person with supporters.

To summarize, the most suitable framework overall is React. The runtime of the three frameworks is pretty similar, but Angular is more designed for bigger applications and not built for small ones what can be seen in the performance: It is slower in startup and has a bigger bundle size than the others. Comparing the frameworks by popularity, React is more asked and seems to be more popular. In addition, a big company is the founder of the framework and these are the reasons for preferring React: More people are using and know React at the moment and to keep maintainability, it is more likely to find a developer, who knows React than one who knows Vue.

Not having to reinvent the wheel, we include libraries for implementing some features: The basic and needed structure of the react application is created with react's CREATE REACT APP[15], an out of the box application creation tool. For building the interactive table we include react-table (version 7.0.0)[16] and for both plots we use react-vis (version 1.11.7)[17]. Also included is react-tabs[18] and JSZip[19]. For testing we choose jest.js[20] as it is served with CREATE REACT APP.

3.2 Structure of Features and Deployment Process

The interactive features in the old version of BENCHEXEC HTML table are all implemented in one HTML file one below each other. As mentioned, one focus of modern frontend frameworks is the component-based structure. React also furthers that so the features are now in different files while developing. Figure 3.1 shows the new structure of the application. CREATE REACT APP separates its sources from other elements of the application e.g. builds and configurations. They are stored in the folder `src` which is also segmented in

[13] Seen on GitHub on August 9th 2019 on `https://github.com/`
[14] Seen on GitHub on August 9th 2019 on `https://www.xing.com/jobs/`
[15] `https://github.com/facebook/create-react-app`
[16] `https://www.npmjs.com/package/react-table`
[17] `https://uber.github.io/react-vis/`
[18] `https://github.com/reactjs/react-tabs`
[19] `https://stuk.github.io/jszip/`
[20] `https://jestjs.i`

components, data, tests and utils. In the folder `src\components` all main features are stored each as one component (thus each as a file). There are five tabs and two overlays. All of them are held and navigated by the Overview.js which also keeps the current state and data of the whole application. Each component also has own states but only concerning the feature itself. In the moment of affecting another part of the application, states or data have to be handed over to the overview. This guarantees a consensus of data in all components. One example is the filtering of the table: With setting a filter the currently shown data is affected. Hence the functions to set and build the filter and how the data is filtered is prepared in the component `reactTable.js` but the function to adapt the data of the application is stored in the `Overview.js`. Every component is shown in Figure 3.1 with its functionalities, used frameworks and dependencies. `App.js` imports the CSS file and `Overview.js` to render the complete application. `Overview.js` is a different file to keep the opportunity of expanding the frontend application and creating a new independent part in parallel `Overview.js` and its components (displayed in green boxes with red outline). As the survey revealed most users first have a look at the summary. Since it is located at the bottom of the table in the current implementation, the users complained about long scrolling. To resolve this issue the summary has been separated from the table and added as a new component respectively its own tab which can be seen in Figure 3.1. In Chapter 4 this topic will be explained in more detail.

The bundle process minifies and bundles all components and style files in one `bundle.min.css` and one `bundle.min.js` (green arrows). Create React App brings along an out of the box bundle process. By default it splits the code into multiple styling and multiple JavaScript files (so called chunks). As we want to include everything into a single template (`template_react.html`) later, the bundling process needs to be adapted accordingly. In the HTML file the chunks are brought together with the data and the final HTML file can be dynamically generated. To generate the HTML file with the new react template the flag `--react` has to be set when using table-generator. The used data is prepared and handed over by the Python script of `table-generator`. It is subsequently parsed to JSON and imported in the `template_react.html` which dynamically generates the final HTML (see Section 3.3).

In comparison to the old version, we now have a deployment process. It can be handled with one command by switching to the folder `react-table` then running `npm run build` for building or `npm run tablegenerator` for building and testing with example content (taken from `components/data/data.json`). The benefit is that the code is now in files per component plus an additional file for styling and a file containing stateless and common used functions. This makes it cleaner and easier to maintain. In addition, the size of the generated

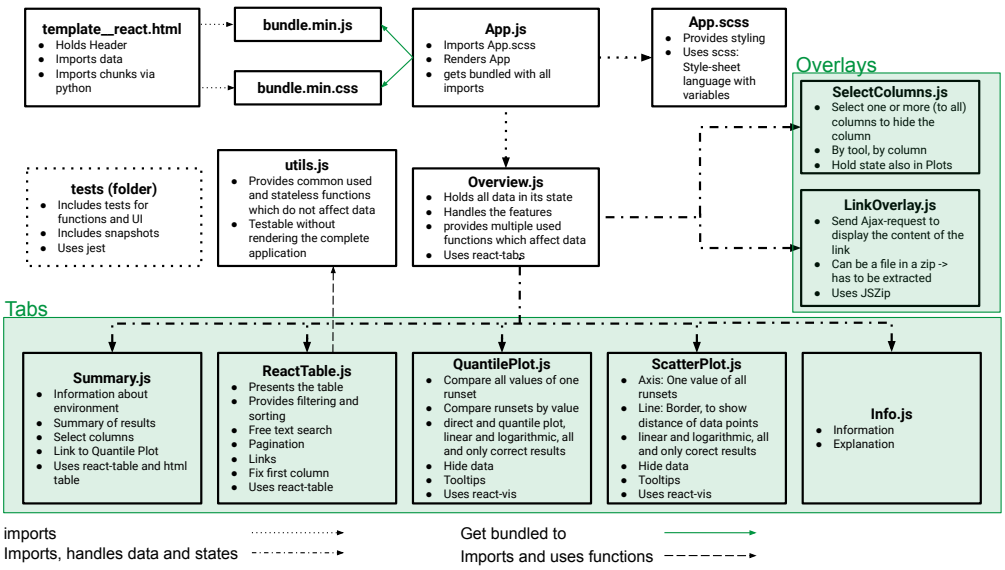


Figure 3.1: Structure of the New HTML Application

HTML is smaller, because of the client-side rendering. With outsourcing some functions into the file `utils.js` components are clearer. In addition automatic testing is easier and faster because to test functions in components requires rendering the whole application (which requires a completely mocked data set). This is the first way of testing. The second is the so called snapshot testing to see if the UI has changed. For both variants an example is implemented and has to be expanded when extending the application. To run automated tests you can use the command `npm run test`. All commands can be adapted in `package.json`

3.3 Data Structure

As the software architecture will be completely new, it is essential to evaluate the data structure and adapt it. So far, `template.html` received its data via two ways: On the one hand the data is directly prerendered by a Python templating engine[21] in the HTML template. On the other hand some data is saved in JavaScript variables to process them via jQuery. As all data will be rendered on the client-side in the future, all the data need to be provided as one object to access them in the react application. This is done in JSON format. JSON is a format which is easy to read, write, parse and generate.

[21] `https://pyrocore.readthedocs.io/en/latest/tempita.html`

It is built in two structures: Arrays (lists in Python) and key/value pairs (dicts in Python). This makes it easy to handover the data from Python script to the template (HTML) and is the basis for data handling of the new implementation[22]. In the new version all needed data is prepared and handed over to `template.html` as JSON objects in the Python script. There, the data is stored in a variable and separated in objects depending on their area of use: Header information, tool/runset information, results per rows, statistics for the summary of the table and properties. The old implementation had some redundant data like the information about columns in each row. Since this information is no longer needed and to reduce the size of the data, it was removed from each row but placed in the information about the tools so every tool has now the information about its columns. One row only provides the results of it including the properties and information about its task. Its results are split as objects for each runset in an array. This unifies the way of passing the data and reduces its size. The results are objects in an array in two variations: Formatted and original. The original is used for comparing and processing with the data (e.g. sorting or calculating the plots), the formatted version is mostly for displaying. In this way we reduce the actions that have to be done in JavaScript. The properties (specifying which parts of the row id shall be displayed) are also placed as one array in the data ('props') because it is always the same for every row. A complete comparison of two rows with the same content but with the old and the new implementation can be found in the appendix B. By generating a table with same data in the new and old version and storing it in an JSON file we can compare the size of both result objects. In the following example the BenchExec result contains 23553 runs. Generating a table with the old version and storing its results in an JSON file, the file's size is 91,1 megabyte. Doing the same process with the same BenchExec result but setting the flag `--react` (so that the new version is used), which adapts the data as described, the size of the JSON file is 47,9 megabyte. This means the file is less than the half of the size as in the old version.

The preparation takes place in `util.py`. The methods `prepare_rows_for_js`, `prepare_run_sets_for_js` and `prepare_stats_for_js` take a copy of the original data (to ensure, that the original data can be used on other places if needed) and returns it in the presented way above. As mentioned the rows lose a lot of their stored data. This is handled with objects to exclude so if more information can be removed in the future work, the keys have only to be written in these removing objects.

[22] `https://www.json.org/`

Chapter 4

Improving the User Interface

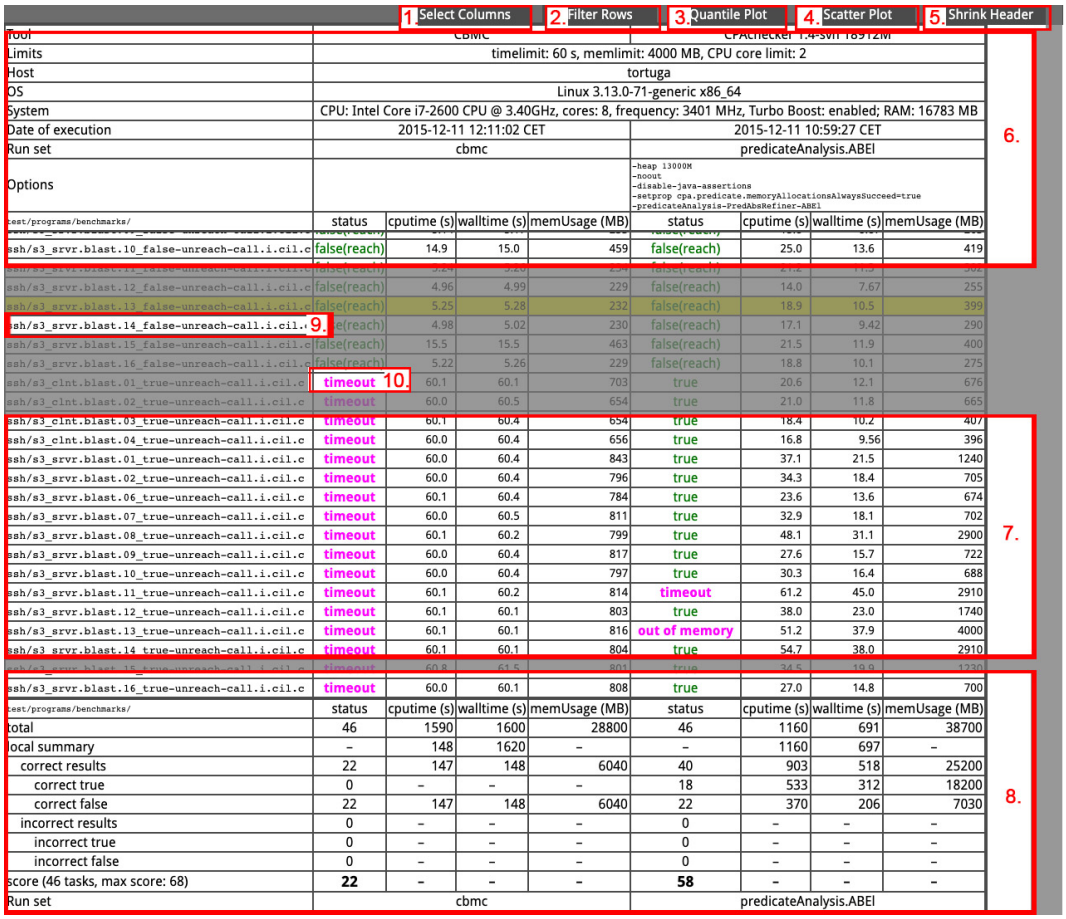
In the following chapter, we attend to the improved user interface (UI) of the BenchExec HTML tables. Before changing it is necessary to analyse, which features are implemented at the current status because - as in the introductions already mentioned - there is no overview or holistic concept of the whole application. After the analyses we focus on the fundamental alterations, difficulties and challenges of improving the UI and connections between features in terms of data and states.

4.1 Analysis of Existing Features

In Figure 4.1 and Figure 4.2 all features or the link to display them is displayed. These are the same images as used in the first survey so the numbering and naming is used in this section, in the survey as well as in the whole work. The following list takes up each characteristic, describes its functionality, features, and coherencies with other characteristics.

(1) *Select columns* (Figure 4.1)
By clicking on this first button in the header, an overlay opens and you can select and deselect the columns of each runset you want to display. It is implemented as a table with green cells for the selected and red cells for the deselected columns. With a click on one cell, it toggles the color and visibility. This also works for all columns of one runset (click on the name of the runset) and all columns of one type for each runsets. You can also open this feature with a click on the header of the filename column. If columns are invisible, they are not shown in the plots, too.

(2) *Filter rows* (Figure 4.1)
This feature is also shown in an overlay and opened by clicking the link. You can select filters out of the columns for each runset (in this example status, cpu time, walltime and memUsage). By choosing the filter a multi-



Tool	CBMC	CPAchecker 1.4-svn 18912M
Limits	timelimit: 60 s, memlimit: 4000 MB, CPU core limit: 2	
Host	tortuga	
OS	Linux 3.13.0-71-generic x86_64	
System	CPU: Intel Core i7-2600 CPU @ 3.40GHz, cores: 8, frequency: 3401 MHz, Turbo Boost: enabled; RAM: 16783 MB	
Date of execution	2015-12-11 12:11:02 CET	2015-12-11 10:59:27 CET
Run set	cbmc	predicateAnalysis.ABEI
Options		-heap 13000M -noout -disable-java-assertions -setprop cpa.predicate.memoryAllocationsAlwaysSucceed=true -predicateAnalysis-PredAbsRefiner-ABEI

test/programs/benchmarks/	status	cputime (s)	walltime (s)	memUsage (MB)	status	cputime (s)	walltime (s)	memUsage (MB)
ssh/s3_srvr.blast.10_false-unreach-call.i.cil.c	false(reach)	14.9	15.0	459	false(reach)	25.0	13.6	419
ssh/s3_srvr.blast.12_false-unreach-call.i.cil.c	false(reach)	4.96	4.99	229	false(reach)	14.0	7.67	255
ssh/s3_srvr.blast.13_false-unreach-call.i.cil.c	false(reach)	5.25	5.28	232	false(reach)	18.9	10.5	399
ssh/s3_srvr.blast.14_false-unreach-call.i.cil.c	false(reach)	4.98	5.02	230	false(reach)	17.1	9.42	290
ssh/s3_srvr.blast.15_false-unreach-call.i.cil.c	false(reach)	15.5	15.5	463	false(reach)	21.5	11.9	400
ssh/s3_srvr.blast.16_false-unreach-call.i.cil.c	false(reach)	5.22	5.26	229	false(reach)	18.8	10.1	275
ssh/s3_clnt.blast.01_true-unreach-call.i.cil.c	timeout	60.1	60.1	703	true	20.6	12.1	676
ssh/s3_clnt.blast.02_true-unreach-call.i.cil.c	timeout	60.0	60.5	654	true	21.0	11.8	665
ssh/s3_clnt.blast.03_true-unreach-call.i.cil.c	timeout	60.1	60.4	654	true	18.4	10.2	407
ssh/s3_clnt.blast.04_true-unreach-call.i.cil.c	timeout	60.0	60.4	656	true	16.8	9.56	396
ssh/s3_srvr.blast.01_true-unreach-call.i.cil.c	timeout	60.0	60.4	843	true	37.1	21.5	1240
ssh/s3_srvr.blast.02_true-unreach-call.i.cil.c	timeout	60.0	60.4	796	true	34.3	18.4	705
ssh/s3_srvr.blast.06_true-unreach-call.i.cil.c	timeout	60.1	60.4	784	true	23.6	13.6	674
ssh/s3_srvr.blast.07_true-unreach-call.i.cil.c	timeout	60.0	60.5	811	true	32.9	18.1	702
ssh/s3_srvr.blast.08_true-unreach-call.i.cil.c	timeout	60.1	60.2	799	true	48.1	31.1	2900
ssh/s3_srvr.blast.09_true-unreach-call.i.cil.c	timeout	60.0	60.4	817	true	27.6	15.7	722
ssh/s3_srvr.blast.10_true-unreach-call.i.cil.c	timeout	60.0	60.4	797	true	30.3	16.4	688
ssh/s3_srvr.blast.11_true-unreach-call.i.cil.c	timeout	60.1	60.2	814	timeout	61.2	45.0	2910
ssh/s3_srvr.blast.12_true-unreach-call.i.cil.c	timeout	60.1	60.1	803	true	38.0	23.0	1740
ssh/s3_srvr.blast.13_true-unreach-call.i.cil.c	timeout	60.1	60.1	816	out of memory	51.2	37.9	4000
ssh/s3_srvr.blast.14_true-unreach-call.i.cil.c	timeout	60.1	60.1	804	true	54.7	38.0	2910
ssh/s3_srvr.blast.15_true-unreach-call.i.cil.c	timeout	60.8	61.5	801	true	34.5	19.9	1230
ssh/s3_srvr.blast.16_true-unreach-call.i.cil.c	timeout	60.0	60.1	808	true	27.0	14.8	700
test/programs/benchmarks/	status	cputime (s)	walltime (s)	memUsage (MB)	status	cputime (s)	walltime (s)	memUsage (MB)
total	46	1590	1600	28800	46	1160	691	38700
local summary	–	148	1620	–	–	1160	697	–
correct results	22	147	148	6040	40	903	518	25200
correct true	0	–	–	–	18	533	312	18200
correct false	22	147	148	6040	22	370	206	7030
incorrect results	0	–	–	–	0	–	–	–
incorrect true	0	–	–	–	0	–	–	–
incorrect false	0	–	–	–	0	–	–	–
score (46 tasks, max score: 68)	**22**	–	–	–	**58**	–	–	–
Run set	cbmc				predicateAnalysis.ABEI			

Figure 4.1: All Features of BenchExec HTML Tables Part 1 - Screenshot taken from `https://sosy-lab.github.io/benchexec/example-table/svcomp-simple-cbmc-cpachecker.table.html`

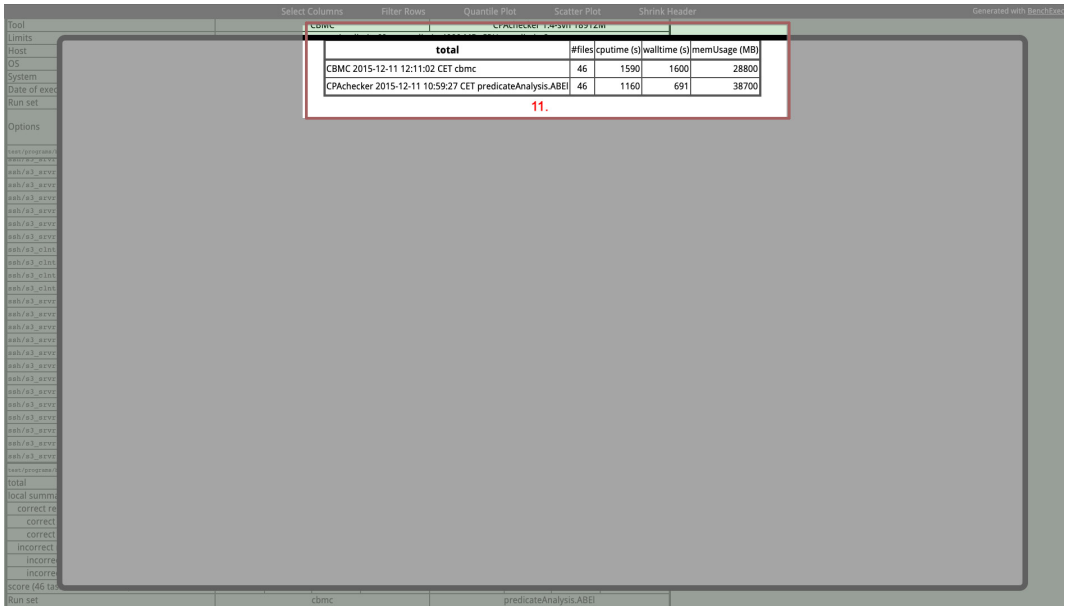


Figure 4.2: All Features of BenchExec HTML Tables Part 2 - Screenshot taken from `https://sosy-lab.github.io/benchexec/example-table/svcomp-simple-cbmc-cpachecker.table.html`

select select field with all values of the this column for the respective runset is shown. From these one or more (by pressing the shift key) values can be chosen to set the range to be displayed. After choosing, the filter can be reset, applied and inverted. It is also possible to select values or none by clicking the respective button. With the click on one button on the right side, the overlay closes and the filters are applied except resetting: then the overlay does not hide.

(3) *Quantile plot* (Figure 4.1)
Quantile plot has two modes and is also shown in an overlay. One mode displays all runsets compared on one parameter (e.g. cputime), the other one visualizes all parameter (columns) of one runset. Each value is represented with a linked dot. In the first and preselected mode, x-axis represents the list of tasks, y-axis its value for the selected parameter (like cputime). On the top you can choose within a dropdown which mode and parameter (column) should be presented (e.g. cputime). Each runset / parameter has one color, the legend is on the right bottom. There you can toggle whether to present the runset(-values) or not and highlighted by hovering. On the bottom of the left hand side there are three buttons to toggle: Direct plot and quantile plot, logarithmic or linear scale and

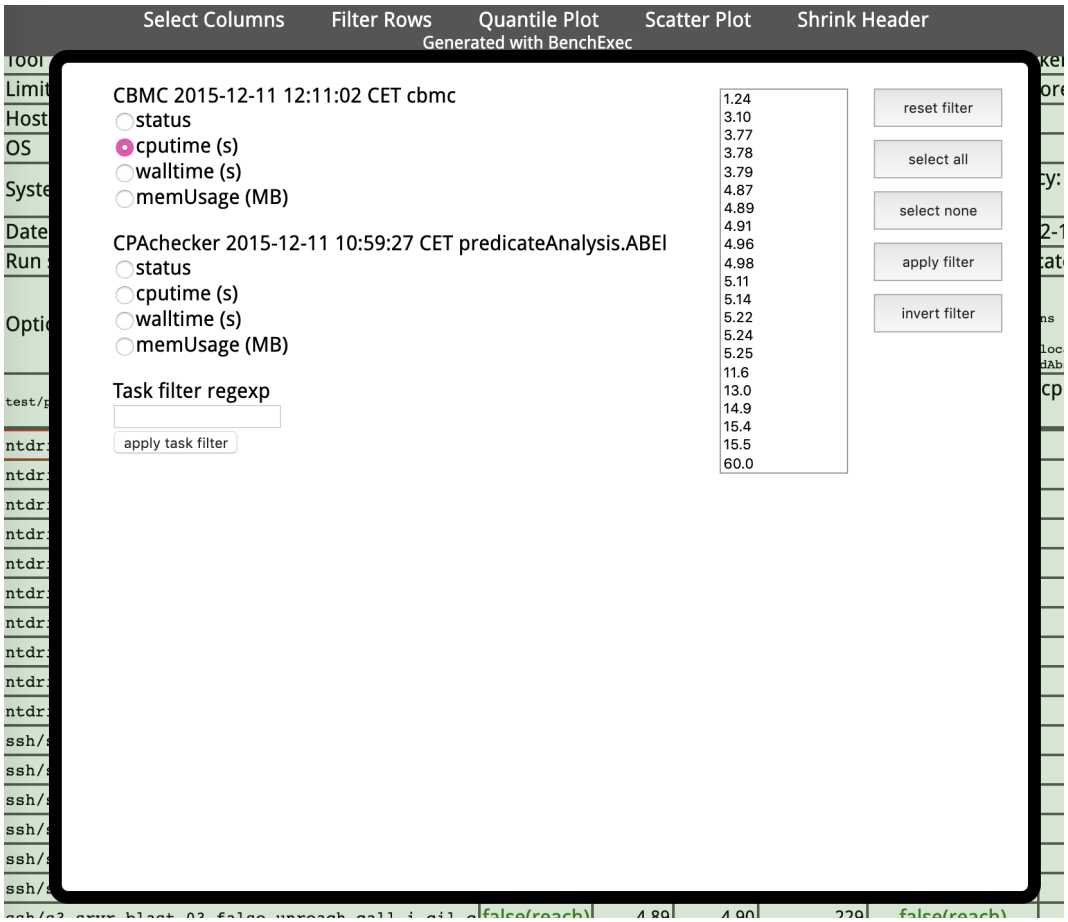


Figure 4.3: The Feature *Filter rows* in the Overlay - Screenshot taken from `https://sosy-lab.github.io/benchexec/example-table/svcomp-simple-cbmc-cpachecker.table.html`

to show all results or only the correct ones. In a direct plot displaying only correct results, the incorrect ones are bypassed (have no point) so the line connects only the correct results visualized by a dot. By hovering on one dot the name of task and the y-value is displayed in a tooltip. If the selected value is of the type text an ordinal y-axis is shown.

(4) *Scattern plot* (Figure 4.1)
The *Scatter plot* is also presented in an overlay. It shows a two dimensional data visualization. Dots represent the values of variables: One variable is the x- and one the y-axis. In a dropdown you can choose which axis should represent which column (like status, cputime, e.g. from which runSet) In a third dropdown you can choose a factor from 2 to 100.000.000 for additional lines in the plot, they have the slope $y = n \cdot x$ and $y = \frac{1}{n \cdot x}$ where n is the factor of the dropdown for additional line mentioned above (default value is 10). This is to see fast, whether there are values more than factor n away. On the bottom of the overlay there are two buttons: One to switch between the presentation of the logarithmic or the linear scale and one to switch whether the user would like to show all or only the correct results. In this case, the plot first filters whether correct or not and then builds the intersection of tasks, in which all runsets the category is "correct". For example this means if you compare two runsets in a task, where for one runset the category is correct but not for the other one, it is not shown in the plot. Axis have not always the same gaps as they fit with the results. With linear scale the red line ($y = x$) hits always the point (0|0) in comparison to the logarithmic scale, because of $\log 0 = undefinded$. For both plots a plugin called jqPlot is used[1].

(5) *Shrink header* (Figure 4.1)
This is the last button in the header. It toggles the display status of *toolinformation* (whether it is displayed or not).

(6) *Toolinformation* (Figure 4.1)
Toolinformation - better called environment information - is on top of the table and can be hidden. This information about the runsets and the environment for the executed benchmarking process are displayed in a given sequence. Cells which are same for both runsets are merged and displayed only once. The cells for "options" are written in Teletype because they are command line options. By clicking on the cells of the row "Run Set" the overlay with the *quantile plot* of this runset opens.

(7) *Result table* (Figure 4.1)
This is the main part of BenchExec HTML tables. It displays all results of the runsets for the given and visible columns. Different runsets are next

[1] `http://www.jqplot.com/index.php`

to each other, one row shows one task. Exact structure and naming is explained in 2.1.4. The values are rounded with significant decimal numbers (which can be adjusted in the TABLE-GENERATOR) and the decimal point of each value of a row has to be directly among the one of the next row to clearly show differences in the size of the value. With a click on the header of a column the quantile plot of this value opens in an overlay.

(8) *Summary* (Figure 4.1)
This section shows a summary of all results from the table. The values are generated via the TABLE-GENERATOR and are not calculated by JavaScript. This includes, that they are static, so if rows are filtered the score is not changing. It shows the number of correct and incorrect results (for each true and false). The summary also shows only the columns which are visible (and not deselected in *Select columns*).

(9) *Link to source/task definition* (Figure 4.1)
Each row represents the values of the results of one task. These tasks have identifications and source codes (definitions). The identification is displayed in this cell, consisting of its name and properties. If the cell is provided with a given link clicking on the cell opens an overlay and the definition / source is requested via AJAX. There are four opportunities to find the asked definition: It can be local or online, resided in a zip or directly behind the linked path. These four options have to be tried to request. If none is successful an error-message is shown.

(10) *Link to tool output* (Figure 4.1)
At the status value of each runset for the specific task the tool output can be linked. The AJAX-request works exactly like the one for the task definition, described above.

(11) *Summary table popUp* (Figure 4.2)
This feature is also shown in an overlay and displays the results of summary in another way than in the table: One row displays the summary results of one row for only one runset (like local summary, correct results etc.). By clicking on the description in the head you can switch between the summary rows. So in every table only the results of one row for all runsets are shown underneath.

(12) *Tooltips*
There are a number of tooltips in different variations: In the table itself, the tooltips are displayed after circa three seconds (on fields with actions) beneath the cursor to explain which action is possible by clicking here. The tooltips of the summary show several additional values like average or minimum for the specific cell. These information are given via the data. In the plots, they have a different appearance like in the table with an opacity and they are shown above the cursor.

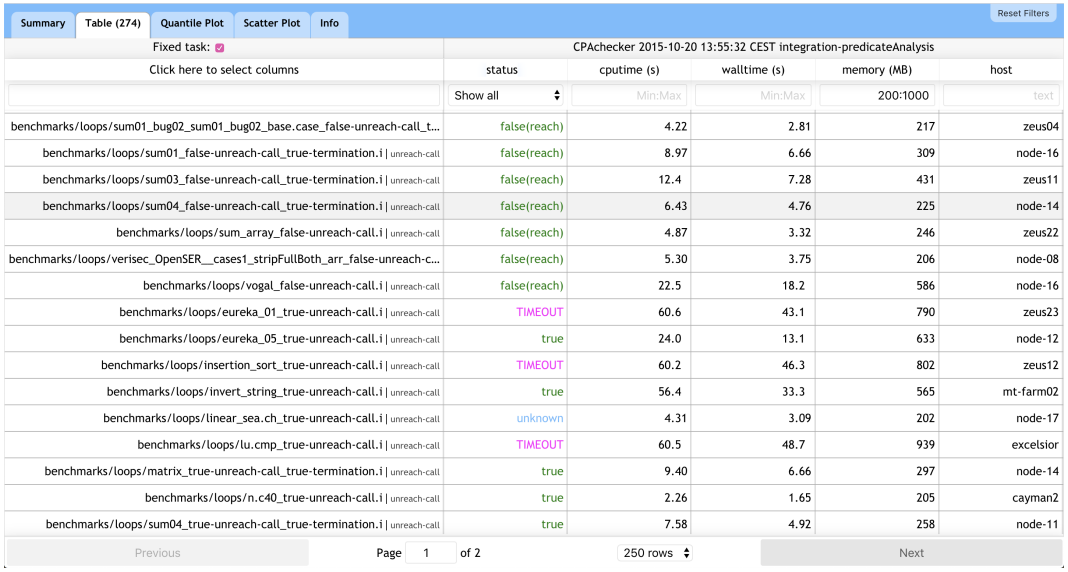


Figure 4.4: Table of the New Version of BenchExec HTML Table with Example Data

4.2 Implementations with Special Challenges and New Approaches

Most features and functionalities were taken over to the new version. No implementation can be inherited because of the removal of jQuery but some concepts of functionalities can be adapted. In the following, only the biggest changes, special challenges and new approaches are described. An example of the new version is shown in Figure 4.4. The result table is displayed in this example and it is filtered by memory (with a minimum of 200 and a maximum of 1000). Additionally the fourth line is hovered.

4.2.1 Tabpanel

The structure of the complete application new: Instead of buttons and their overlays a tab panel on the top of the application is shown. You do not have to open and close overlays any more but can switch between functionalities with only one click and see where you are in the application. Thus the space for content is bigger. In the tab of the table (which is shown in an example in Figure 4.4) the number of displayed rows (depending on the applied filters, also shown in Figure 4.4) is shown. If one or more filters are set, it is possible to reset them by the button in the tab panel from everywhere. So the filters

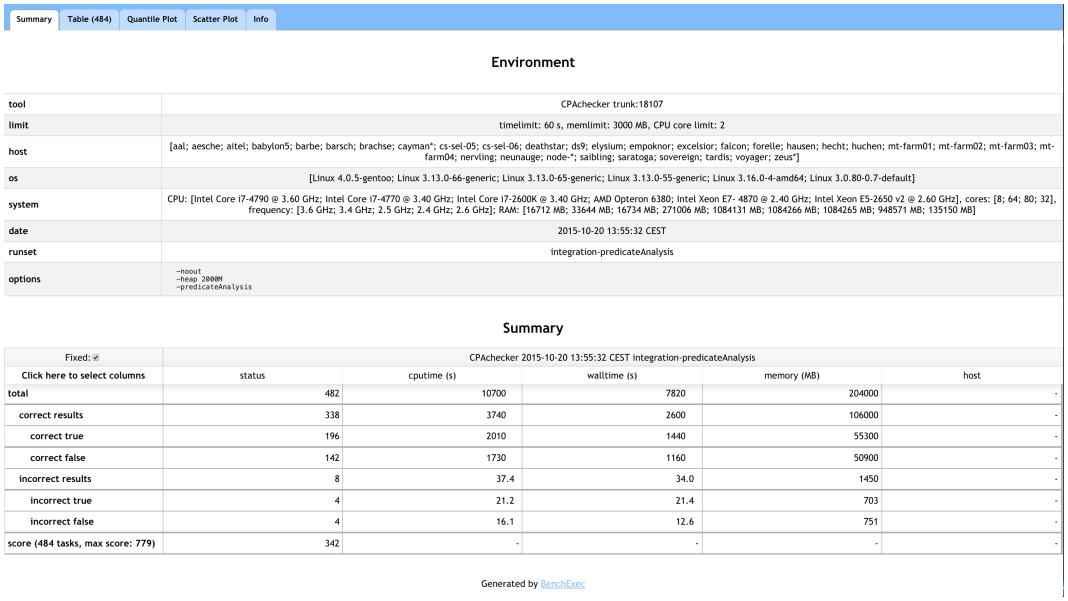


Figure 4.5: Summary of the New Version of BenchExec HTML Table with Example Data

can be reset e.g. while having a look on the plot so the difference between filtered and non-filtered data is more obvious. This is one step to reach the goal of creating an application, that is easy to handle and every feature and information is easy to reach. For the background of the tab panel the blue of the Software-system lab is used. To get a continuity it is defined as the key color. This means, that it is used for every highlight in the whole application like links or important buttons.

4.2.2 Summary and Information

By opening the application you see a summary of information first as the shown example in Figure 4.5. This includes the data of used environments (like tools, limits, options and so on). Below, the summary of data is presented. It shall solve the problem of long scrolling to see the summary, which was criticized by many participants. Now the first look on the summary can be done completely without waiting. It also replaces the button for shrinking the header because now the environment information is separated from the table and more space can be used to show content of the table.

4.2.3 Table Implementation

The mentioned space is now used to add a bar for filtering: Each column now has the option to filter without any overlay or long lists of filters. There are three types of filters: A selection field for the status where you can select a category or a concrete value, an input field for text and an input field for values. The value can be a concrete number, a minimum, a maximum or a combination of it. For example, if users want to find all tasks where the cputime is less than 3 seconds but higher than 1 second they can put '1:3' in the input field over the column cputime. This is represented through a placeholder. To prevent filtering after every typed character a delay is set to 500ms. All set filters are displayed in the input field so they are always reachable in this tab. Filtering is part of the used library *react-table* but the filter itself as well as the filter function have to be overwritten with a custom filtering because of many reasons: setting a placeholder, different handling of filtering the types of values in the column (text or number), the selection field for filtering by status, the way data is structured and the explained approach of typing minimum or maximum. This is handled with a regular expression.

In this filtering method the mentioned distinction from Section 3.3 is needed: To decide if the row has to be returned by minimum or maximum (with `>=` and `<=`) the original value has to be used to get no rounding problems. If users enter a concrete value they are searching for, the formatted value has to be searched because this is what the user sees in the table. By clicking the table head of a column the rows are sorted. The out of the box sorting function also is overwritten by a custom sorting. Horizontal scrolling is now possible for the whole table or for only the results. The column with task names can be fixed through a checkbox. On the bottom of table a pagination is implemented. On the left and right side, there are buttons to show the previous or next page with results, in the middle a select field is placed to set up the number of shown results per page and a field to go to a specific page. Sorting and filtering works for all pages and not only for the shown one.

Chapter 5

Evaluation

The last part of this thesis is about ascertaining if implementing a new version of BenchExec HTML tables brought improvements for the users and if the requirements from Section 2.5 are fulfilled. For this we use the results of the first survey and compare them with answers from another one which was conducted after the implementation. The second survey was provided as a Google form for eleven days. It was publicized through an email to the users of BenchExec and a link on the BenchExec's GitHub page (this was the same publication as the first survey only three days less time to participate). The full results of each survey with the list of questions can be found in Appendix C and D.

5.1 Comparison of the Surveys

The users were advised to play around with the new application and attend to get the information that they are interested in when normally using the HTML tables. The first survey was answered by 16 and the second one by 13 people. Overall 91.3% of the participants believed that the new version was an improvement in comparison to the old table.

Comparison of Ratings of BenchExec HTML Tables In one section of both surveys, the participants could rate usability, design and user-experience of HTML tables of BenchExec with a number from one to five. In the first question (which asked about the usability) one was set as "I need a lot of time to find the feature/information I am searching for" and five was set as "every feature/tool is reachable and easy to find", similar were the other ratings: One was the most negative rating and five the best one. The average rating for all three asked points raised from the old to new version: Usability raised from 3.2 in the old version to 4.4 in the new one, the design from 3.3 in the old

to 3.4 in the new version and the rating of user experience raised from 3.0 in the old version to 4.1 in the new one. In the second survey, the participants could also rate three things in direct comparison between old and new version. The first of these questions asked, if the users need now more or less time (in comparison to the old version) to find the information they are searching for (also with one as the worst and five as the best rating). In this case, twelve out of the 13 answered that they are faster (or much faster) in finding their searched information (average: 4.4). Only one participant answered with a rating of two out of five Also the next question was answered positive from the view of the new version: The velocity of loading with the new version was asked in comparison to the old one. In this case, nine out of thirteen people answered that loading is much faster now (average: 4.5), two people answered that it is faster and two participants rated it with three out of five. The last question (in direct comparison) was about the velocity of interacting with the table (like clicking, sorting, e.g.). Eleven out of thirteen rated it with "faster" or "much faster" (average: 4.5), two people choose the middle.

In sum, the participants rated the new version in every question better than the old one so it is an improvement for them to work with the new application. In the first three questions this is shown by comparing the average: The rating's average of all three questions is higher for the newer version. On the second three questions, an average beneath three would support the old version but for all questions it is higher than 4.3.

Comparison of Usability for Every Feature in Detail A question about the usability of the features in detail was asked in both surveys, so we can compare seven features directly for the new and the old version: *Scatter* and *quantile plot*, the *table* itself, *filter rows*, *select columns*, the *summary* of results and the *tool information*. Some features were asked about only in one survey: *Shrink header* was not mentioned in the second survey, as it was removed in the new version because of the restructuring. *Link to source code* and *tool output* were not asked again because the feature is inherited but only with a new (and updated library) and the *summary table pop up*, because it was used by so few people that a new solution has to be found (explained in Chapter 6 conclusion). In the second survey there are some new features like *fixing the tasks* (on the left hand side), to *sort rows*, the *pagination* and the *information tab* so they can not be compared with the old version. In chart 5.1 we can see a higher average of usability of the new version of BenchExec HTML tables for every feature. The average was calculated without the answers "never used". In the old version there are only two features with an average rating more than four: *Quantile plot* and *Select Column*. No average of the new version's features is less than four, the ratings of *table itself* and *scatter*

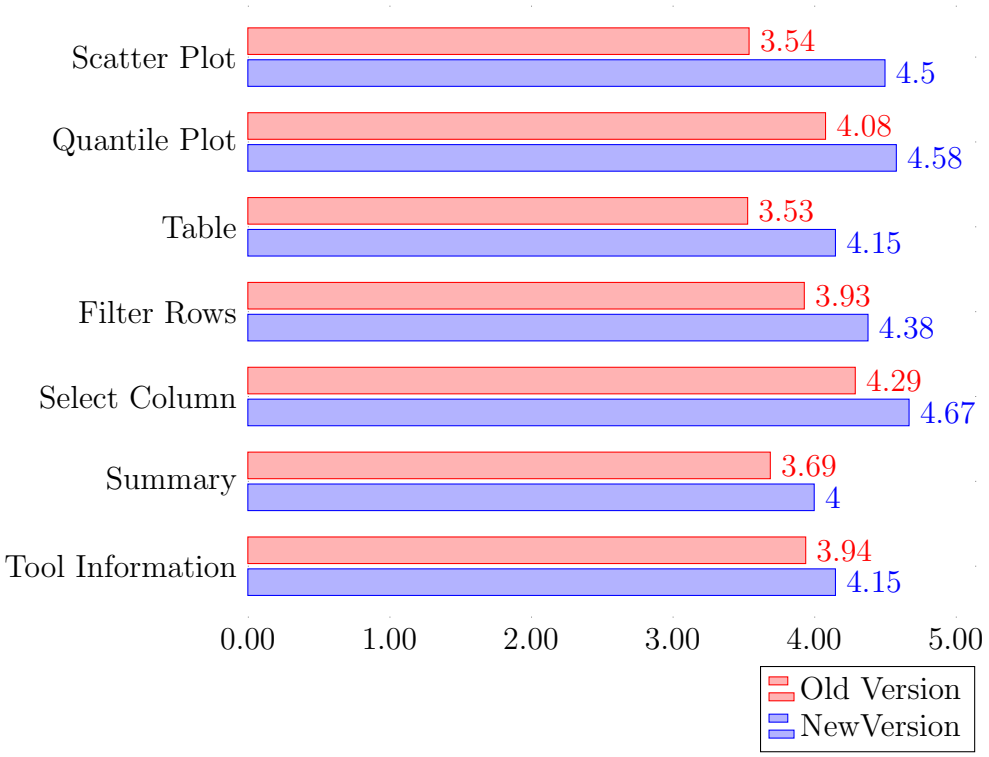


Figure 5.1: Comparison of Average Usability for Every Feature in Detail

plot increased most (the average valuation of the *table* increased by 0.53, the average valuation of *scatter plot* increased by 0.96). The ratings of the new version of comparable features are visualized in Figure 5.2. To mention is that the *filtering* of rows was rated with "very bad" from one person. Nine participants out of thirteen rated it with "very good", three with "good", one with "ok" besides the one with "very bad". This can be associated with an answer in the last part of the survey number two where the participants had the chance to tell their additional comments. In more than one questions there were answers about the *filtering*: They said, that the capabilities were more advanced in the old version and that the table looses a lot of value because there is no chance for boolean combinations of *filtering*. Another point to improve is the summary: Eight participants out of 13 rated it with "good" and three with "very good" but there is one person, who rated it with bad. It also has the worst average (four which means "good") which can be seen in Figure 5.1. The *tool information* has its lower quartile and lower whisker at three and its upper quartile and upper whisker at five. There is no outlier, six participants rated it with five, three with four and four with five. All of the other features except the summary have their upper whisker in the upper quartile. Especially *quantile Plot* has only ratings between four and five. This shows, that most of the participants of the survey are satisfied with the new implementation of features. The average of all of the features of the new version are four or greater in comparison to the old version. The features of the old version with the highest average is *quantile plot* with 4.08.

5.2 Evaluation of Final Comments in Second Survey

In the last part of the second survey, there were six open questions where the user had space to write down their experiences and opinions. They were asked about improvements, regressions, missing features and bugs. In general many participants were thankful and praised the work. The most mentioned thing was the velocity: In total, there were eleven positive comments about the speed of plots, loading or performance of the whole application. Some complaints were remedied after closing the survey, like the visibility of lines between columns, bold titles of tabs, the contrast of toggle buttons in the *select columns* overlay and the styling of buttons. Also the missed information about the possibility to fix the first column was added. Some written points were not changed because their comments were only subjective. For example two users complain about the font-size: One thinks that it is to small while the other one recommends to set it smaller "to fit more content on screen". One point

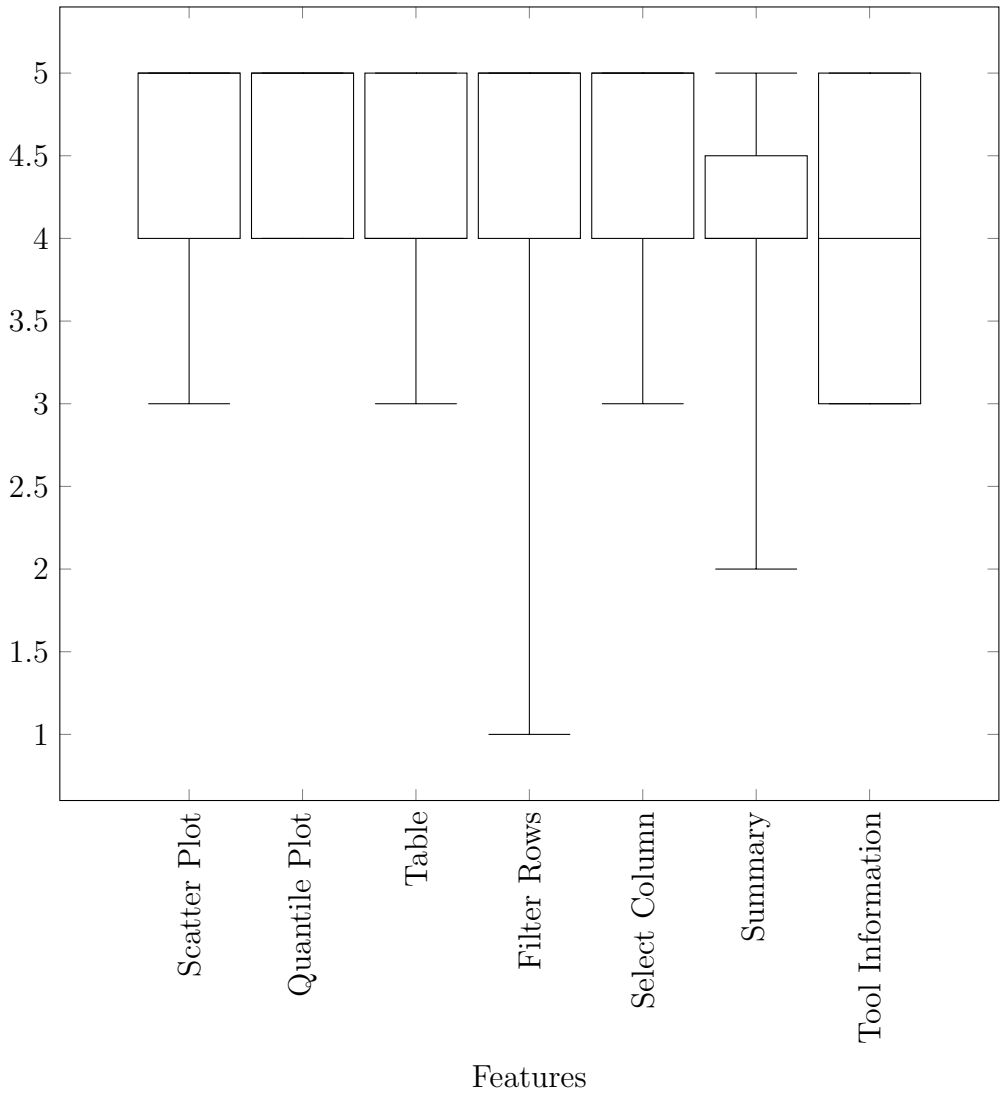


Figure 5.2: Box Plot of Results for Comparable Features of the New Version

with lots of opinions is the *filtering*. On one hand, the participants like the new approach of it, especially that the set filters are displayed on top of the table. On the other hand, they want it "more advanced" with the possibility of boolean combinations. This is a bigger issue which could be rethought, evaluated and implemented in another work. One other point is the separation of runsets: One participant complained that it is hard to distinguish between different runsets. Since this is only mentioned once, the design of the table could be overthought and tested by users but it is not indispensable because the average meaning about the usability of the table was good.

5.3 Fulfilment of Requirements

According to Section 2.5 and with regards to the second survey we verify in this section, if these requirements are fulfilled with the new version of BenchExec HTML tables. With the flag `--react` (Section 3.2) in the tool table-generator the new interactive table of benchmarking results, which is build with the modern JS library react, can now be generated. It provides all features of the old version except the *summary table pop up* which was nearly never used according to the survey. Some features are available now in a completely new manner like the filtering but always visible while working with the result table. Even though, the structure of the application is new and the usage of some features is different, the users have no problems to find them (e.g. "splitting parts (*summary*, *table* etc.) into different tabs" as the biggest improvement). They voted all comparable functionalities more positive in the new version than the old one. This is reinforced by 92,3% of the participants of the second survey who answered that they needed less time to find the information they were interested in. Based on the aforementioned information, it is clear that the application is intuitively usable and the users achieve their goals effectively and efficient. The new application now provides functionalities like *sorting* and *pagination* which was not given before and additionally mentioned in the area for comments of the second survey (e.g. pagination the biggest improvement). The performance is positively mentioned eleven times in the open comments part of the second survey. In the questions about the velocity and time for loading, interacting and finding information, the averages of answers is always higher than 4. Time for interacting and loading is never voted with 'much slower' or "slower". What the user could not vote for was the maintainability and clear structure of the code. During development the frontend is now clear structured and separated in files per feature and implemented with a common used and famous framework so it should be easy for developers to find the section of code they are looking for.

Chapter 6

Conclusion and Perspective

As one of the survey participants wrote as an answer, implementing a new architecture and improving the UI was the first step to create a comprehensive fast interactive application to visualize the results of BenchExec HTML tables. Finding and implementing the new framework with its new structure took most of the time and resources. Since most of the features are needed by the users and the framework is completely new the features had to be implemented in a new way. Another difficulty was that there was no holistic concept or a full list of features and (hidden) functionalities which popped up while implementing something else. This shall be solved with this thesis but made it hard to estimate the effort for implementing and caused bugs and challenges. The user-centered approach and the coherent first survey helped a lot to understand the real users, their needs and ways of using the application. This revealed the biggest problem of long loading times which shall be solved with the new version. The application can now be extended and further developed with a solid and holistic basis. One point to expand is the filtering. This was already rethought, improved and positively voted for but since there are user needs which are not fulfilled it may be expanded by, for example, boolean combinations or the performance and the filtering concept could be improved and evaluated. The filter section could also be presented in the plot tabs so the users have the possibility to filter their graphical presentation of results directly in the plot. Smaller points to improve are to switch off the pagination or to provide the possibility to download the plots as `SVGs`. Another huge point to improve is the summary in combination with the never found feature *summary table pop up*. As it was not possible to implement a whole new and configurable dashboard and the focus on features which are used more often, the separate summary can be seen as the minimum viable product. Next steps could be to calculate all summary results via JavaScript which would make it possible to show also a summary of filtered results or

expand it (e.g. Survey answer: "perhaps cut-off when there are only 1% or so of results of some type"). This would make it configurable and would give the possibility to create a handy dashboard which combines *summary*, *summary table pop up* and kind of a dashboard.

The new version of BenchExec HTML tables should not be considered final: As the human-centered design approach says it should be an iterative process which means, that the users should be regularly included in an continuous process of improving and extending the application while never losing sight of the big picture.

List of Figures

List of Tables

Bibliography

[1] David Benyon. *Designing interactive systems: A comprehensive guide to HCI and interaction design*. Pearson Edinburgh, 2010.

[2] Dirk Beyer, Stefan Löwe, and Philipp Wendler. Reliable benchmarking: Requirements and solutions. *International Journal on Software Tools for Technology Transfer (STTT)*, 21(1):1–29, 2019.

[3] Peter Bühler, Patrick Schlaich, and Dominik Sinner. *JavaScript*, pages 32–53. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018.

[4] ISO. *Ergonomics of human-system interaction – Part 210: Human-centred design for interactive systems (ISO 9241-210:2010); German version EN ISO 9241-210:2010*. Beuth Verlag, Berlin, 2011.

[5] J. Lumsden. *Handbook of research on user interface design and evaluation for mobile technology*. Number Bd. 1 in Handbook of Research on User Interface Design and Evaluation for Mobile Technology. Information Science Reference, 2008.

[6] Bernhard Preim and Raimund Dachselt. *Interaktive systeme: band 1: grundlagen, graphical user interfaces, informationsvisualisierung*. Springer-Verlag, 2010.

[7] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. Starexec: a cross-community infrastructure for logic solving. In *International joint conference on automated reasoning*, pages 367–373. Springer, 2014.

[8] Philipp Wendler. Towards practical predicate analysis. PhD Thesis, University of Passau, Software Systems Lab, 2017.

Appendix A

Comparison of Frameworks in Detail

Name Duration for...	angular-v7.1.4-non-keyed	react-v16.6.0-non-keyed	vue-v2.6.2-non-keyed
create rows creating 1,000 rows	153.8 ± 3.3 (1.00)	168.9 ± 6.2 (1.10)	162.1 ± 6.7 (1.05)
replace all rows updating all 1,000 rows (5 warmup runs).	36.6 ± 2.7 (1.00)	41.7 ± 0.7 (1.14)	46.2 ± 1.0 (1.26)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	203.6 ± 6.9 (1.00)	244.9 ± 13.6 (1.20)	328.5 ± 15.0 (1.61)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	53.4 ± 5.5 (1.00)	54.4 ± 2.1 (1.02)	206.4 ± 37.2 (3.87)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	38.3 ± 2.6 (1.00)	38.6 ± 1.8 (1.01)	69.6 ± 1.8 (1.82)
remove row removing one row. (5 warmup runs).	33.2 ± 1.8 (1.00)	38.2 ± 0.8 (1.15)	44.1 ± 1.2 (1.33)
create many rows creating 10,000 rows	1,500.3 ± 15.1 (1.08)	1,794.0 ± 49.9 (1.29)	1,387.1 ± 49.3 (1.00)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown	364.2 ± 9.9 (1.00)	367.9 ± 6.8 (1.01)	372.1 ± 7.1 (1.02)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown	392.3 ± 11.9 (1.93)	202.8 ± 7.2 (1.00)	235.9 ± 7.6 (1.16)
slowdown geometric mean	1.09	1.10	1.42

Name	angular-v7.1.4-non-keyed	react-v16.6.0-non-keyed	vue-v2.6.2-non-keyed
ready memory Memory usage after page load.	5.2 ± 0.2 (2.46)	2.3 ± 0.2 (1.11)	2.1 ± 0.0 (1.00)
run memory Memory usage after adding 1000 rows.	9.4 ± 0.0 (1.38)	6.8 ± 0.0 (1.00)	7.0 ± 0.0 (1.03)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	9.7 ± 0.0 (1.32)	7.9 ± 0.0 (1.07)	7.4 ± 0.0 (1.00)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	9.5 ± 0.0 (1.33)	10.8 ± 0.1 (1.52)	7.1 ± 0.0 (1.00)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	6.8 ± 0.0 (1.83)	4.6 ± 0.0 (1.26)	3.7 ± 0.0 (1.00)
slowdown geometric mean	1.61	1.18	1.01

Name	angular-v7.1.4-non-keyed	react-v16.6.0-non-keyed	vue-v2.6.2-non-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	2,969.9 ± 0.2 (1.36)	2,327.2 ± 0.1 (1.07)	2,176.9 ± 0.3 (1.00)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	156.3 ± 1.6 (9.77)	26.0 ± 19.5 (1.62)	16.0 ± 0.0 (1.00)
main thread work cost total amount of time spent doing work on the main thread. includes style/layout/etc.	692.9 ± 30.5 (1.18)	591.6 ± 5.2 (1.01)	588.4 ± 9.5 (1.00)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	358.8 ± 0.0 (1.70)	252.3 ± 0.0 (1.20)	211.0 ± 0.0 (1.00)
slowdown geometric mean	2.27	1.20	1.00

Appendix B

Comparison of One Row in Old and New Version

```
{
  "expected_results": {
    "unreach-label": [
      false,
      null
    ]
  },
  "filename": "example_filename",
  "has_sourcefile": true,
  "id": [
    "example_id",
    "unreach-label",
    null
  ],
  "properties": [],
  "results": [
    {
      "category": "wrong",
      "columns": [
        {
          "display_title": null,
          "href": null,
          "number_of_significant_digits": null,
          "pattern": null,
          "relevant_for_diff": false,
          "scale_factor": 1,
          "source_unit": null,
          "title": "status",
          "type":
            {
              "_type": 5,
              "name": "main_status"
            },
          "unit": null
        },
        { ... }, // similiar structure as above
        { ... } // similiar structure as above
      ],
      "columns_relevant_for_diff": [
```

```
40            "status"
41          ],
42          "log_file": "examplelog_file",
43          "score": -32,
44          "sourcefiles_exist": true,
45          "status": "done",
46          "task_id": [
47            "example_id",
48            "unreach-label",
49            null
50          ],
51          "values": [
52            "true",
53            "0.6026s",
54            "2170"
55          ]
56        }
57      ],
58      "short_filename": "example_shortfilename"
59    }
```

```
1     {
2       "expected_results": {
3         "unreach-label": [
4           false,
5           null
6         ]
7       },
8       "filename": "example_filename",
9       "has_sourcefile": true,
10      "href": "example_path",
11      "id": [
12        "example_id",
13        "unreach-label",
14        null
15      ],
16      "results": [
17        {
18          "category": "wrong",
19          "columns_relevant_for_diff": [
20            "status"
21          ],
22          "href": "example_href",
23          "log_file": "examplelog_file",
24          "score": -32,
25          "values": [
26            {
27              "formatted": "true",
28              "original": "true"
29            },
30            { ... }, // similiar structure as above
31            { ... } // similiar structure as above
32          ]
33        }
34      ],
35      "short_filename": "example_shortfilename"
36    },
```

Appendix C

Survey No. One

User Experience of the BenchExec HTML Tables

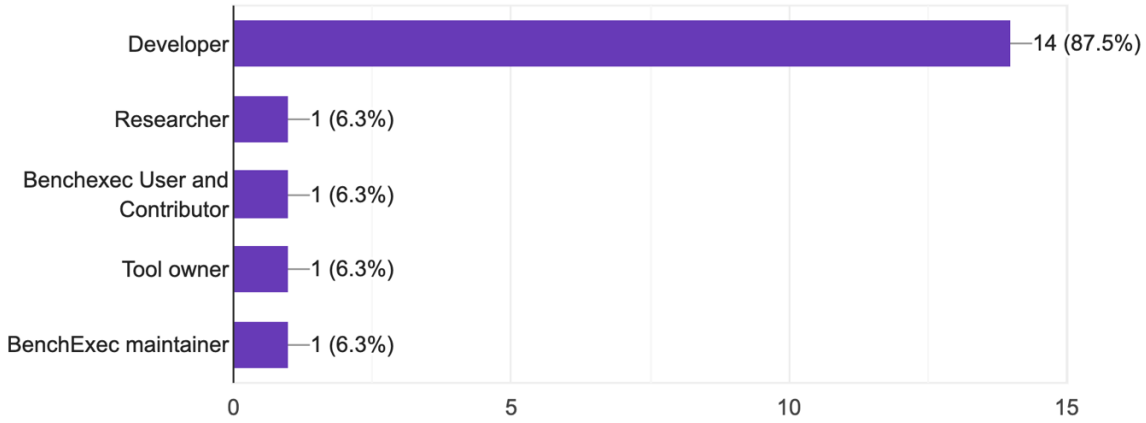
Thank you for participating. As part of my thesis, I want to improve the usability of the BenchExec HTML Tables. With this survey I would like to obtain information from you as the real user of it about your needs, usage and wishes to rise your user experience of BenchExec HTML Tables.

*Required

1. I am a… * (Multiple choice)

- [ ] Developer
- [ ] Other: _____

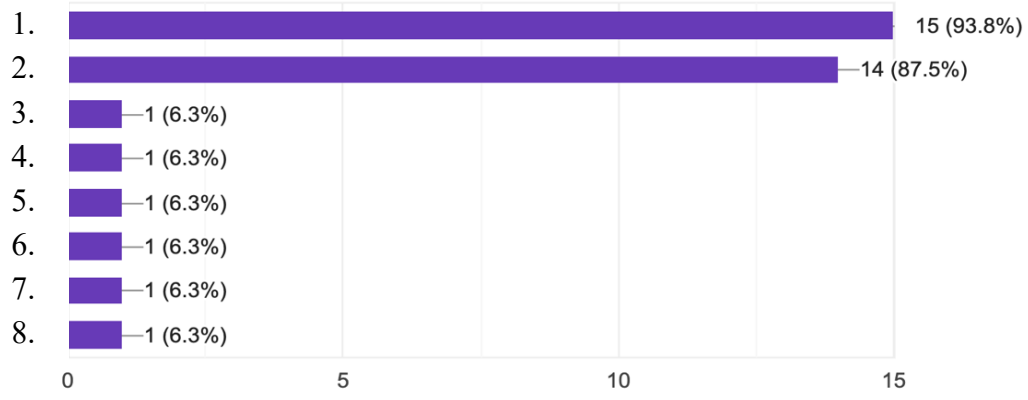
Answers:



2. What do you use BenchExec for? * (Multiple choice)

- [ ] Compare different versions of one tool
- [ ] Compare different tools
- [ ] Other: _____

Answers:



Legende:

1.	Compare different versions of one tool
2.	Compare different tools
3.	extract statistics
4.	Reliably measure/limit/kill a single tool
5.	Make it easy to integrate/adapt tools into framework (through tool-info modules)
6.	Run one tool in parallel
7.	Identify weaknesses of a configuration (tool), compare statistics of different tasks
8.	Execute generic experiments

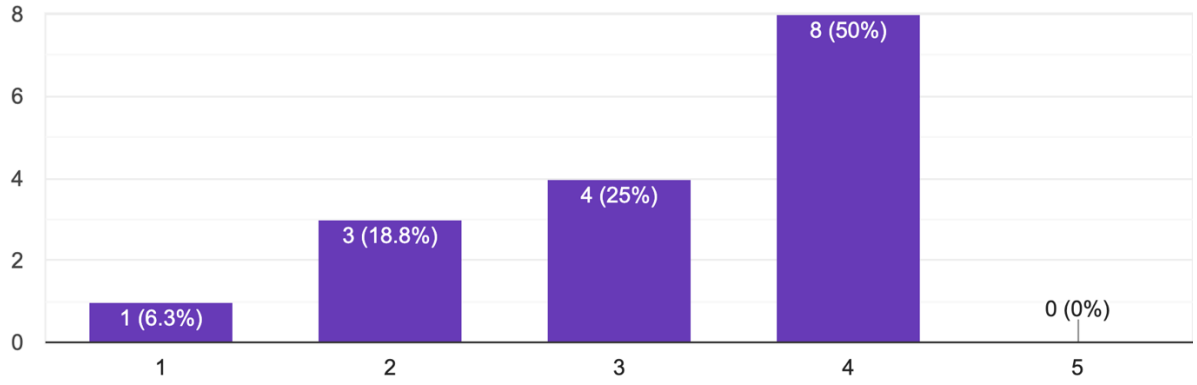
Usability of the BenchExec HTML Tables

In this part, please tell me your opinion about the current HTML Tables. BenchExec HTML Tables means the result list itself as well as all the features.

3. How would you rate the usability of the tables? * (Single choice)

	1	2	3	4	5	
I need a lot of time to find the feature/information I am searching for	◯	◯	◯	◯	◯	Every feature/tool is reachable and easy to find

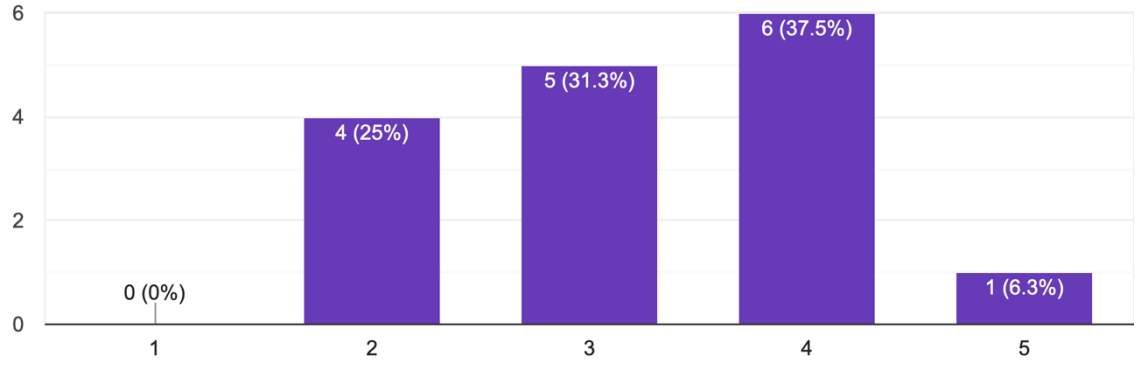
Answers:



4. How would you rate the design of the tables? * (Single choice)

	1	2	3	4	5	
Not at all attractive	◯	◯	◯	◯	◯	Very attractive

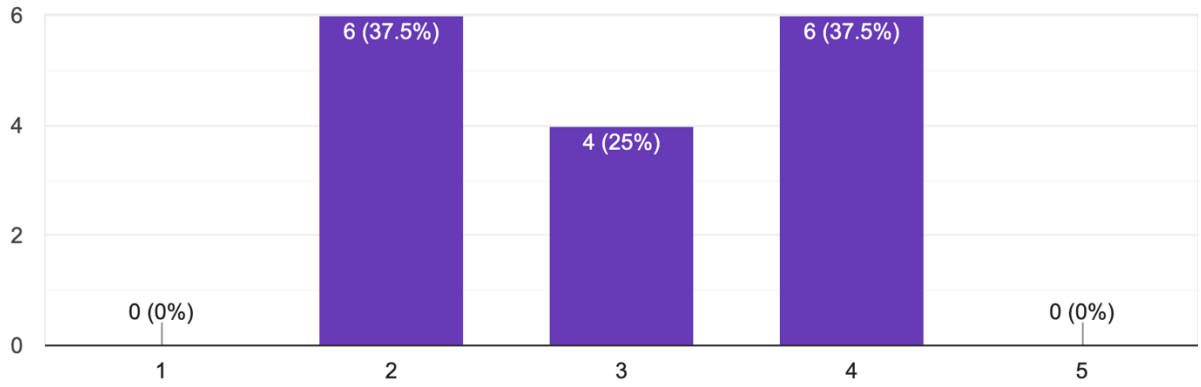
Answers:



5. How would you rate the user-experience of the tables? * (Single choice)

	1	2	3	4	5	
I don't like working with the tables	◯	◯	◯	◯	◯	I like working with the tables

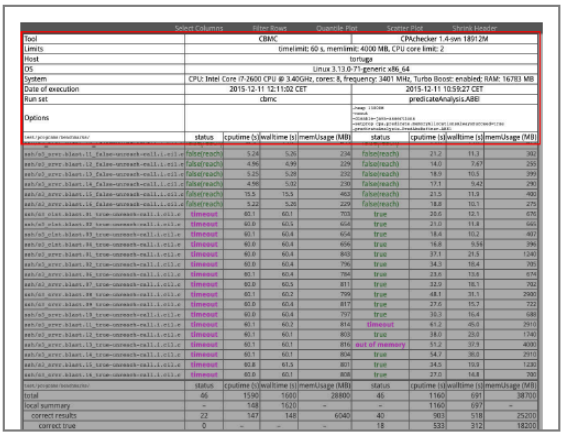
Answers:



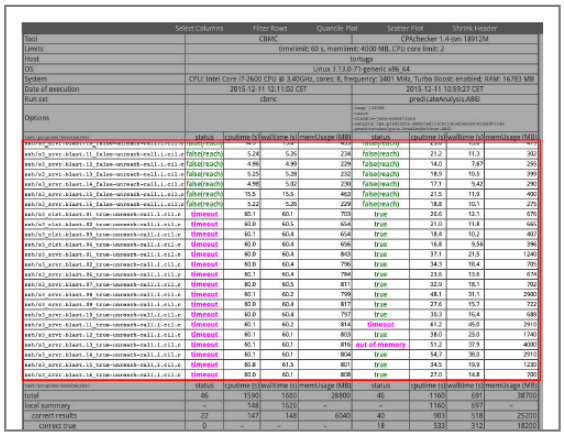
Usage of BenchExec HTML Tables

The following section asks you about your normal usage of BenchExec HTML Tables to get an idea, which features are important and good or bad to handle.

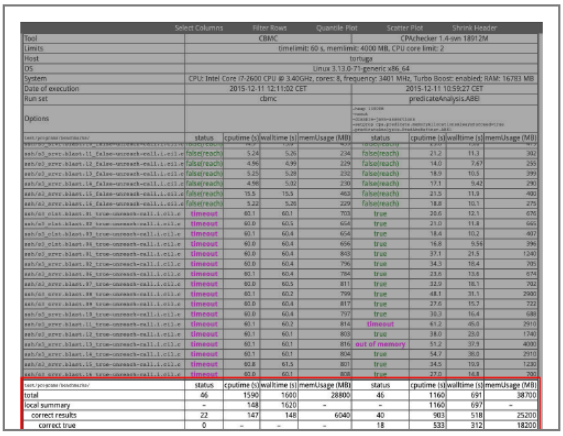
6. When you open BenchExec HTML Tables, what is the first thing you are looking at? * (Multiple choice)



☐ Tool information



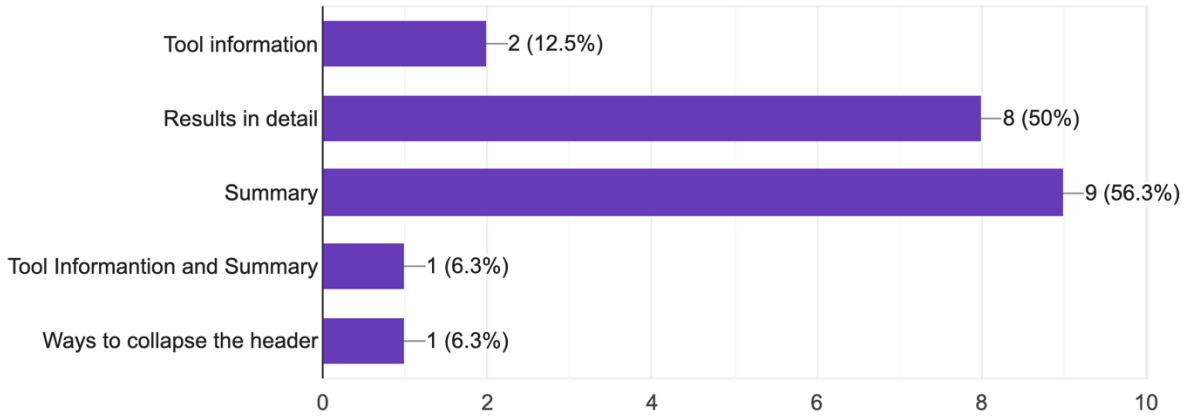
☐ Results in detail



☐ Summary

☐ Other: _____

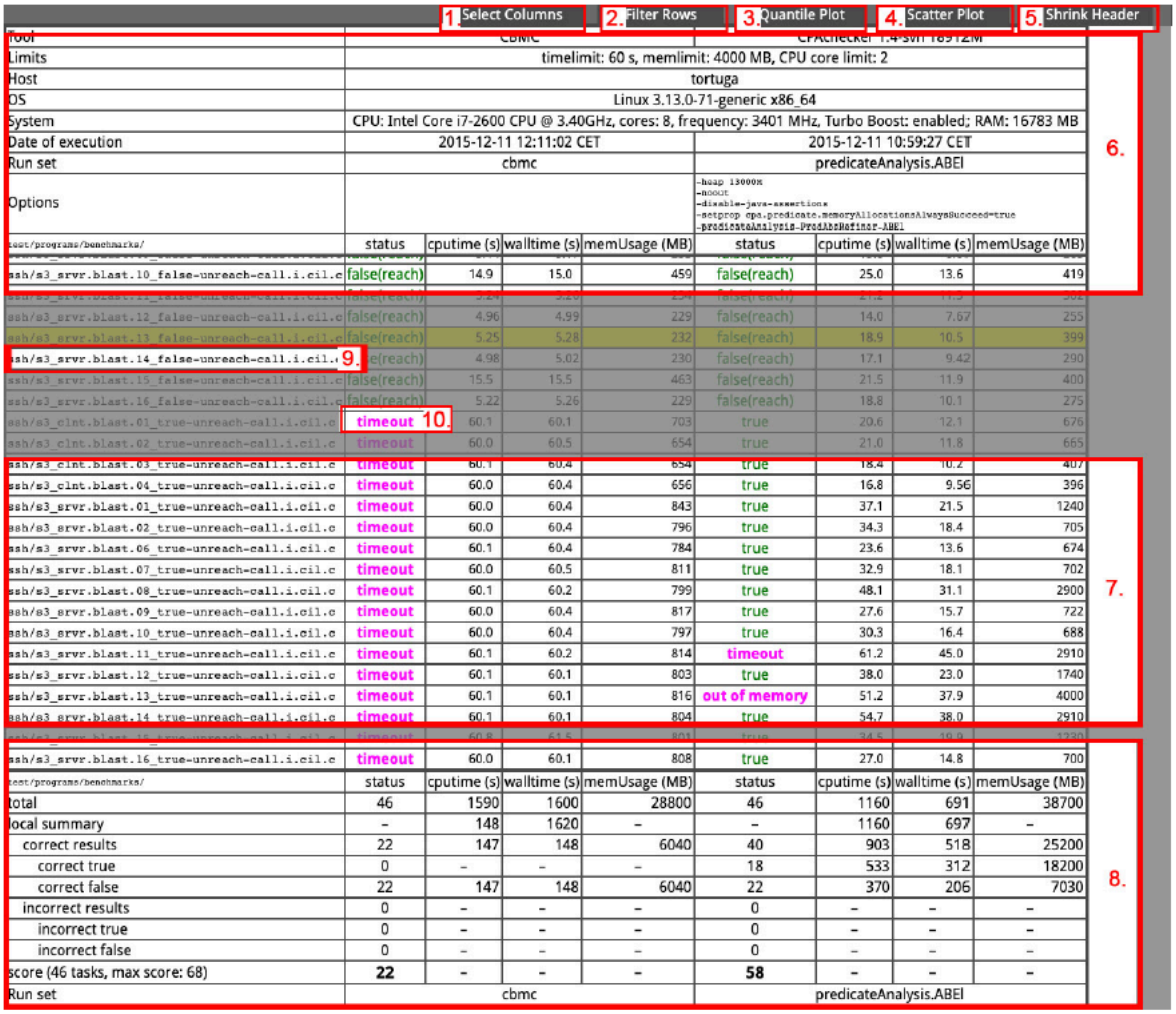
Answers:



Features

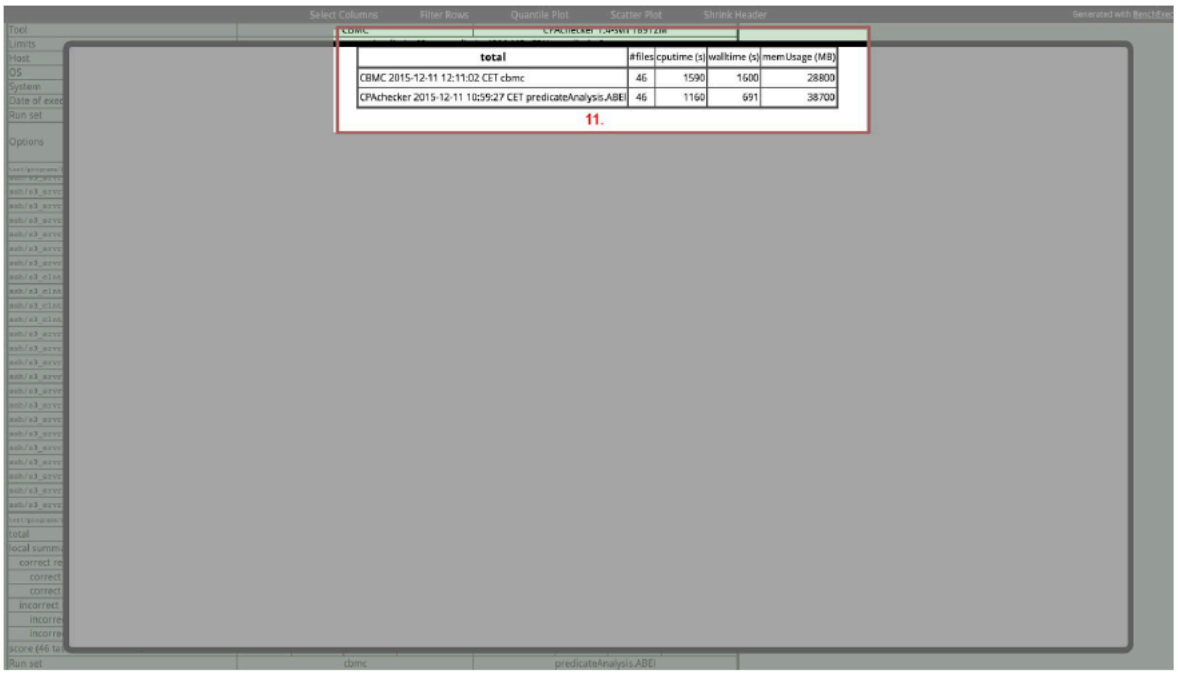
For the next two questions you can use the Feature-Overview below if you do not know, which features are meant.

Features-Overview



Tool	CBMC	CPAchecker 1.4-svn 18912M
Limits	timelimit: 60 s, memlimit: 4000 MB, CPU core limit: 2	
Host	tortuga	
OS	Linux 3.13.0-71-generic x86_64	
System	CPU: Intel Core i7-2600 CPU @ 3.40GHz, cores: 8, frequency: 3401 MHz, Turbo Boost: enabled; RAM: 16783 MB	
Date of execution	2015-12-11 12:11:02 CET	2015-12-11 10:59:27 CET
Run set	cbmc	predicateAnalysis.ABEI
Options		-heap 13000M -noout -disable-java-assertions -setprop cpa.predicate.memoryAllocationsAlwaysSucceed=true -predicateAnalysis-PredAbsRefiner-ABEI

test/programs/benchmarks/	status	cputime (s)	walltime (s)	memUsage (MB)	status	cputime (s)	walltime (s)	memUsage (MB)
ssh/s3_srvr.blast.10_false-unreach-call.i.cil.c	false(reach)	14.9	15.0	459	false(reach)	25.0	13.6	419
ssh/s3_srvr.blast.12_false-unreach-call.i.cil.c	false(reach)	4.96	4.99	229	false(reach)	14.0	7.67	255
ssh/s3_srvr.blast.13_false-unreach-call.i.cil.c	false(reach)	5.25	5.28	232	false(reach)	18.9	10.5	399
ssh/s3_srvr.blast.14_false-unreach-call.i.cil.c	false(reach)	4.98	5.02	230	false(reach)	17.1	9.42	290
ssh/s3_srvr.blast.15_false-unreach-call.i.cil.c	false(reach)	15.5	15.5	463	false(reach)	21.5	11.9	400
ssh/s3_srvr.blast.16_false-unreach-call.i.cil.c	false(reach)	5.22	5.26	229	false(reach)	18.8	10.1	275
ssh/s3_clnt.blast.01_true-unreach-call.i.cil.c	timeout	60.1	60.1	703	true	20.6	12.1	676
ssh/s3_clnt.blast.02_true-unreach-call.i.cil.c	timeout	60.0	60.5	654	true	21.0	11.8	665
ssh/s3_clnt.blast.03_true-unreach-call.i.cil.c	timeout	60.1	60.4	654	true	18.4	10.2	407
ssh/s3_clnt.blast.04_true-unreach-call.i.cil.c	timeout	60.0	60.4	656	true	16.8	9.56	396
ssh/s3_srvr.blast.01_true-unreach-call.i.cil.c	timeout	60.0	60.4	843	true	37.1	21.5	1240
ssh/s3_srvr.blast.02_true-unreach-call.i.cil.c	timeout	60.0	60.4	796	true	34.3	18.4	705
ssh/s3_srvr.blast.06_true-unreach-call.i.cil.c	timeout	60.1	60.4	784	true	23.6	13.6	674
ssh/s3_srvr.blast.07_true-unreach-call.i.cil.c	timeout	60.0	60.5	811	true	32.9	18.1	702
ssh/s3_srvr.blast.08_true-unreach-call.i.cil.c	timeout	60.1	60.2	799	true	48.1	31.1	2900
ssh/s3_srvr.blast.09_true-unreach-call.i.cil.c	timeout	60.0	60.4	817	true	27.6	15.7	722
ssh/s3_srvr.blast.10_true-unreach-call.i.cil.c	timeout	60.0	60.4	797	true	30.3	16.4	688
ssh/s3_srvr.blast.11_true-unreach-call.i.cil.c	timeout	60.1	60.2	814	timeout	61.2	45.0	2910
ssh/s3_srvr.blast.12_true-unreach-call.i.cil.c	timeout	60.1	60.1	803	true	38.0	23.0	1740
ssh/s3_srvr.blast.13_true-unreach-call.i.cil.c	timeout	60.1	60.1	816	out of memory	51.2	37.9	4000
ssh/s3_srvr.blast.14_true-unreach-call.i.cil.c	timeout	60.1	60.1	804	true	54.7	38.0	2910
ssh/s3_srvr.blast.15_true-unreach-call.i.cil.c	timeout	60.8	61.5	801	true	34.5	19.8	1230
ssh/s3_srvr.blast.16_true-unreach-call.i.cil.c	timeout	60.0	60.1	808	true	27.0	14.8	700
test/programs/benchmarks/	status	cputime (s)	walltime (s)	memUsage (MB)	status	cputime (s)	walltime (s)	memUsage (MB)
total	46	1590	1600	28800	46	1160	691	38700
local summary	-	148	1620	-	-	1160	697	-
correct results	22	147	148	6040	40	903	518	25200
correct true	0	-	-	-	18	533	312	18200
correct false	22	147	148	6040	22	370	206	7030
incorrect results	0	-	-	-	0	-	-	-
incorrect true	0	-	-	-	0	-	-	-
incorrect false	0	-	-	-	0	-	-	-
score (46 tasks, max score: 68)	**22**	-	-	-	**58**	-	-	-
Run set	cbmc				predicateAnalysis.ABEI			

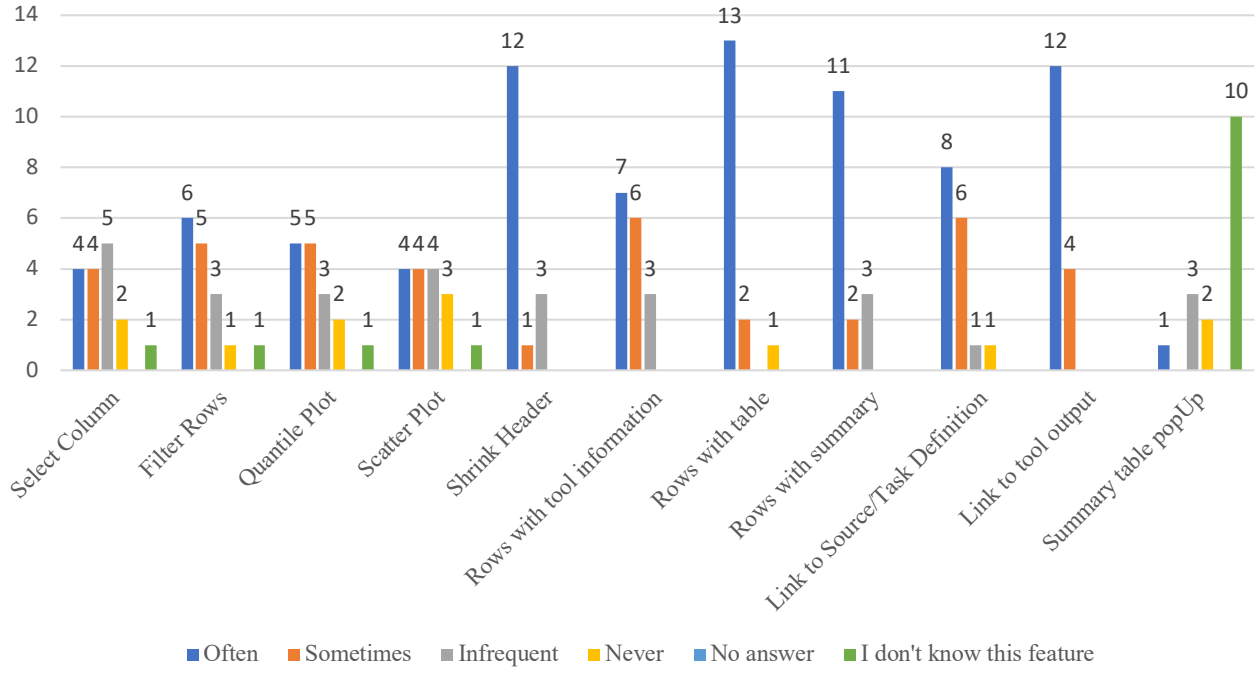


total	#files	cputime (s)	walltime (s)	memUsage (MB)
CBMC 2015-12-11 12:11:02 CET cbmc	46	1590	1600	28800
CPAchecker 2015-12-11 10:59:27 CET predicateAnalysis.ABEI	46	1160	691	38700

7. How often do you use the following features/information? * (Single choice)

	Often	Sometimes	Infrequent	Never	No answer	I don't know this feature
1. Select Columns						
2. Filter Rows						
3. Quantile Plot						
4. Scattern Plot						
5. Shrink Header						
6. Rows with tool information						
7. Rows with table						
8. Rows with summary						
9. Link to Source/Task Definition						
10. Link to tool output						
11. Summary table popUp						

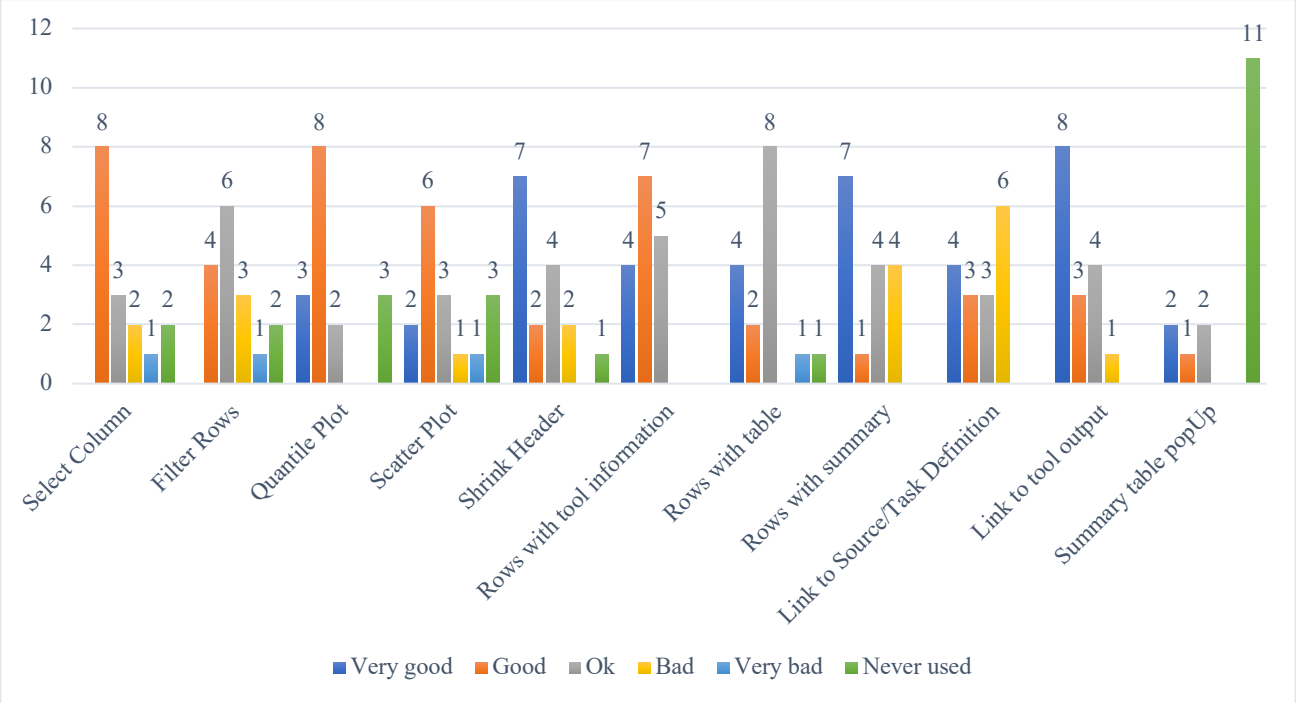
Answers:



8. How food can the following functions be used? * (Single choice)

	Very good	Good	Ok	Bad	Very bad	Never used
1. Select Columns						
2. Filter Rows						
3. Quantile Plot						
4. Scattern Plot						
5. Shrink Header						
6. Rows with tool information						
7. Rows with table						
8. Rows with summary						
9. Link to source code						
10. Link to tool output						
11. Summary table popUp						

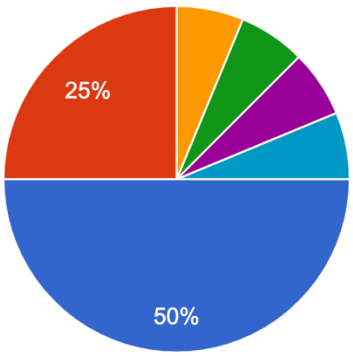
Answers:



9. Do you think a dashboard with a selection of information would help you to get the most crucial results? * (Single choice)

- ◯ yes *Skip to question 11.*
- ◯ no
- ◯ Other: _____

Answers:



- yes
- no
- Do not quite understand the question: what dashboard do you mean?
- Maybe, but I don't think that's a priority.
- My problem has mostly been just being able to scroll through all the result rows--- the tool information header takes up way too much space and sometimes the website is not easily scroll-able.
- I'm not sure I understand the question. Selection of what information?

Dashboard

10. Why do you think a dashboard would not help you to get the most crucial results? (Free text)

Answers:

The table is already a kind of dashboard to me. I would not know which information apart from that I would need. A dashboard might make sense if you want to get an overview over a collection of tables, but not for an individual table. Maybe I did not understand the question correctly.
Again an unclear question
I'm looking at individual benchmarks, not aggregated results
Because often I don't even know what the most crucial results are going to be.
The most crucial results are use case dependent.
Didn't understand the question.

11. Why do you think a dashboard would not help you to get the most crucial results? (Free text)

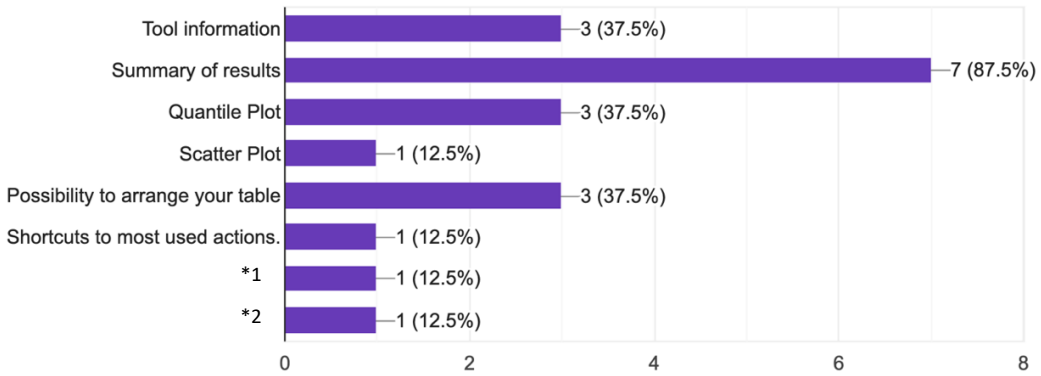
Answers:

In most cases look at the CPUtime quantile plot or filter for exceptions. Shortcuts for these actions would be nice. And viewing the dahsboard while loading large tables in the background would be nice, because the table loads really slow for larger benchmarks.
Depending on the information, it would concisely show me the info I'm interested in
It would allow me to configure my own rating scheme and perhaps allow me to include my own columns and queries
If it is customisable to some degree I can get the information I need on the first glance without having to scroll/click any further (Scrolling/clicking is a time sink).

12. What do you want to have on your dashboard? * (Multiple choice)

- [ ] Tool information
- [ ] Summary of results
- [ ] Quantile Plot
- [ ] Scatter Plot
- [ ] Possibility to arrange your table
- [ ] Other: _____

Answers:



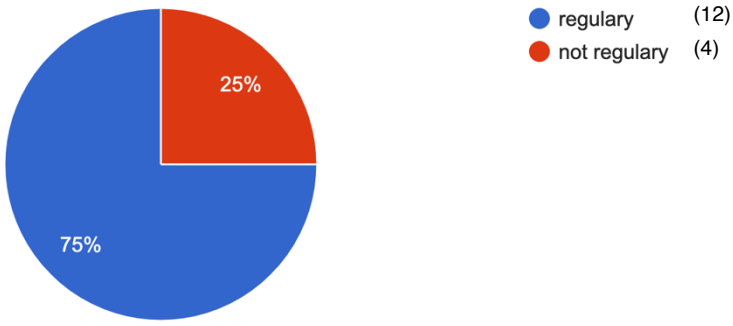
*1) Possibility to arrange your table, A summary that is not just an arbitrary score but also counting the number of true, false, etc. -- essentially numbers for the different status types.
*2) Diff between columns (diff table)

Regularity of usage

13. Do you use the BenchExec HTML tables regulary or just rarely? * (Single choice)

- ◯ regulary *Skip to question 14.*
- ◯ not regulary *Skip to question 15.*

Answers:



If you use BenchExec HTML Tables regulary:

14. How often do you use BenchExec HTML Tables? (Single choice)

- ◯ Once a week
- ◯ one to three times a month
- ◯ Six to eleven times a year
- ◯ One to five times a year
- ◯ Other: _____

Answers:

If you use BenchExec HTML Tables not regulary:

15. How often have you used BenchExec HTML Tables? (Free text)

Answers:

10-20 times
Time to time, when there is a need to estimate new option or configuration
~10 times
When I need to compare results different tools.

16. In which situations do you use BenchExec HTML Tables? (Free text)

Answers:

during Test-Comp
While comparing different CPAchecker configurations
To check whether my tool was working for TestComp
Mostly during competitions.

Final comments

In the end of this survey, you have some space to write down any hints, wishes, suggestions or ideas that would help to improve your own usage.

17. What is the most annoying feature and why? (Free text)

Answers:

long loading time

Deselecting columns takes a long time, because the table is updated on every column I deselect, but e.g. for SV-COMP I often want to deselect all but one column.

first loading the whole table, with summary at the bottom

Compiling time: it takes a lot of time to show the whole report, if there are a lot of benchmarks.

Larger data loads really slow (slow startup, slow plots, slow column selection). The link to the real C-sources is missing since sv-benchmark-repo was restructured (only yml-files are shown, but not the C-files or properties).

filter rows; doesn't work, regex broken, etc

No feature itself is annoying

Large tables load really slowly, in particular with java script

The summary is in parts incomprehensible (I don't even know what some lines mean) and in other cases the need to scroll left and right to match a row description to a number makes it very hard to use.

Long loading times, long response time when filtering or selecting rows/columns when working with large tables.

That it was hard to see the main rows of the table, and that the header took up so much space.

When data is large, getting to summary becomes difficult. I need to scroll a lot and often browser becomes irresponsive.

The tables are very slow. If you filter rows, the summary numbers does not reflect the filtering.

summary is at the bottom: need to scroll and often loading takes a while

loading time

18. What is the most important feature and why? (Free text)

Answers:

statistics, because I want to get a coarse idea of the results

filter rows by result type (filter wrong, filter true etc.)

shrink header, sometimes header is too large for small monitors

Summary and filters, because this is what usually needed for comparison two configurations

2 features: showing quantile plots and filtering rows.

filter rows; essential for drilling down

Scatter and quantile plots, because they give a good overall impression of the data in the tables

linking to tool output and benchmark output , colors

The quick access to a broad range of data points to judge the status of a tool, regressions, performance.

Filtering and selection of entries to focus on information of current interest

Links to output --- mostly to debug how my tool was failing to run in benchexec conditions

Summary of results. It gives quick view result summary.

Quick compare results for different versions of the tool

Tool output, plots, its easy to generate the tables.

Quantile plots, diff tables: to quickly get a picture of the difference between columns

19. What would improve your usage of BenchExec HTML Tables? (Free text)

Answers:

super-fast loading, and a dashboard with summary information

If they were not so darn slow when loading (especially big tables like SV-COMP)

load summary first and stick it to top of the table

Light version will be great, which will contain less information, but will open much faster.

please merge https://github.com/sosy-lab/benchexec/pull/419 or provide a better fix as soon as possible.

provide working implementations for filtering rows according to benchmark outcome, name, etc

* Faster loading times and plot creation,
* If the task name in the very left would stay visible, even when scrolling to the right in large tables,
* Possibility to click on points in scatter plot to get to the corresponding table row (at the moment, it only shows the name which is also hard to copy; so one has to remember the name, close the scatter plot, and use text search to get to the corresponding row)
* Possibility to save or download created plots (e.g. as images, SVGs or pyplot/gnuplot scripts)

A configurable summary view

A machine-processable version. That is _not_ XML. CSV, JSON, Pandas data files.

Better interactive response times

Reordering of features such that most important info comes first followed by details.

Speeding up the rendering, showing plots on the filtered results only, showing summary for the filtered results. Possibility to divide the rows according to categories (.set files).

Scatter plot: use CPU time of rightmost two columns as default

20. What else would you like to get improved?

Answers:

Tables load js from the internet, which means I cannot use them fully if I am somewhere without access to the internet (this however happens very rarely nowadays, as internet access is possible almost from everywhere)

header auto-shrinked

Large tables should load much faster. Maybe compress all internal data and provide the table content lazily after loading the rest.

visual appeal

Loading speed and Java script usage; better way of self-hosting without relying on CDNs or libraries loaded from anywhere in the web

A fixed first column when scrolling left<->right (see also earlier comment). AND: Thank you for setting up this survey and working on this!!!

better filtering capabilities

The source output goes to the .yml file, it would be nice to have an option to go the the actual source file. The interface for filtering tasks is quite uncomfy: you cannot select several criteria at once, the list of statuses has a fixed width and longer statuses cannot be read. Maybe it would be nice to have 'apply' button for selecting columns so that one could adjust all columns before the changed table is rendered (this is tied to the speed of rendering: when the table is re-rendered after every click on a column, it can get very lengthy when un/selecting multiple columns). Some shortcuts for filtering like "different results" (I know about the diff table, but...).

Shrink header by default, but keep tool name

dynamic update of summary rows, better filters, dynamic diff mode

Appendix D

Survey No. Two

User experience of BenchExec HTML tables 2.0

Thank you for participating. As mentioned in the last survey, in my thesis I want to improve the usability of BenchExec HTML tables. With this follow-up I want to find out, if there is an improvement for you in using the new version of HTML tables and what the next possible step could be.

*Required

Instructions

First, please have a look at the tables:

https://www.cip.ifi.lmu.de/~wendler/benchexec_evaluation/00046.-r31603_integration-nightly-induction.2019-07-29_2200.results.html
https://www.cip.ifi.lmu.de/~wendler/benchexec_evaluation/results.2019-07-31_1717.table.html

Use the tables in a manner that is typical for you, e.g. attempt to get the information that you are interested in and play around with the various features.

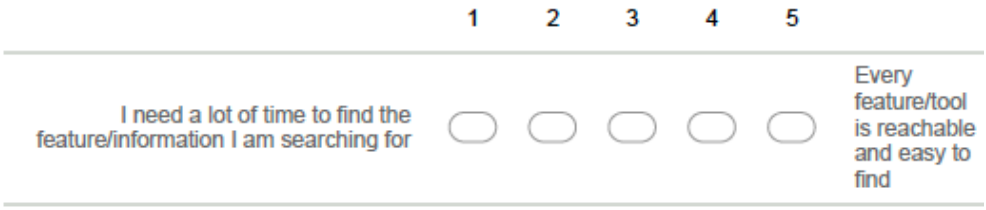
You can also try out generating the table locally, for this:

- check out the branch 'react-table' from https://github.com/sosy-lab/benchexec
- run the table-generator for your runSet(s) with bin/table-generator <yourFile>.xml.bz2 and the flag --react (the command would be bin/table-generator --react <yourFile>.xml.bz2)
- open the generated file in the browser of your choice

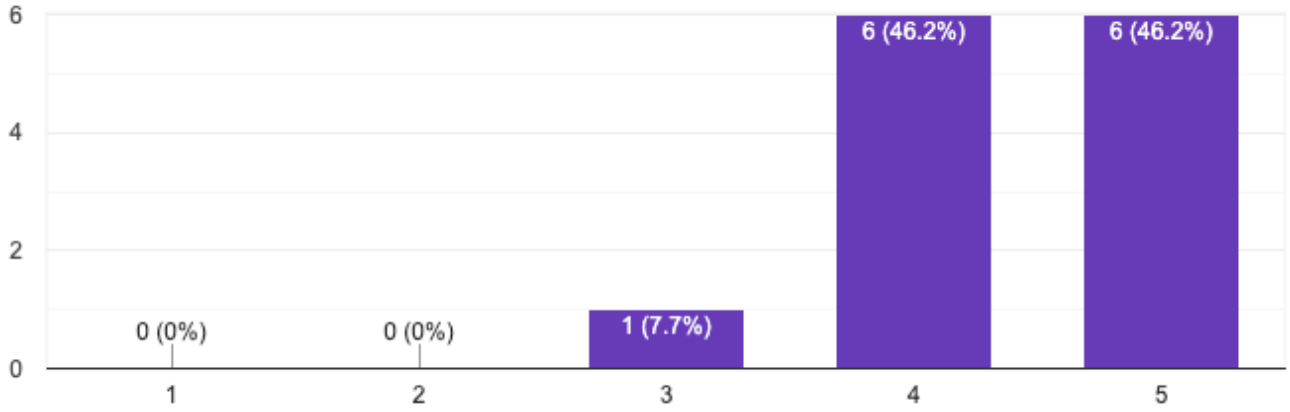
Usability of BenchExec HTML Tables 2.0

In this part, please tell me your opinion about the new HTML tables application. This includes the result list itself as well as all the features.

1. How would you rate the usability of the table application? * (Single choice)

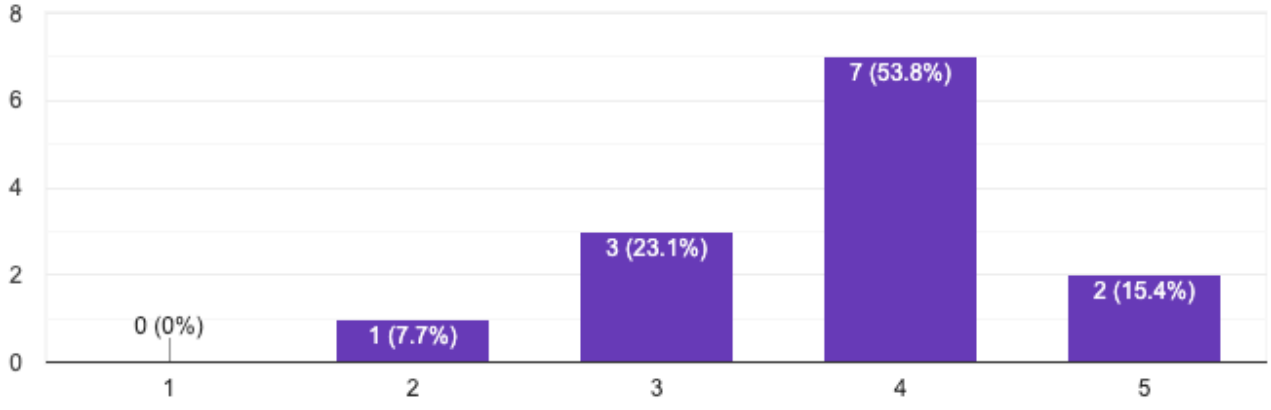


Answers:

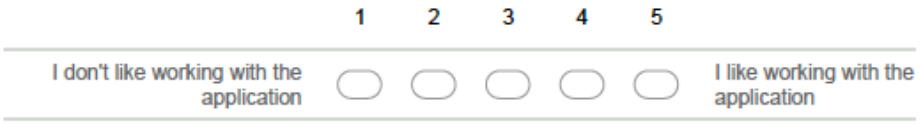


2. How would you rate the design of the application? * (Single choice)

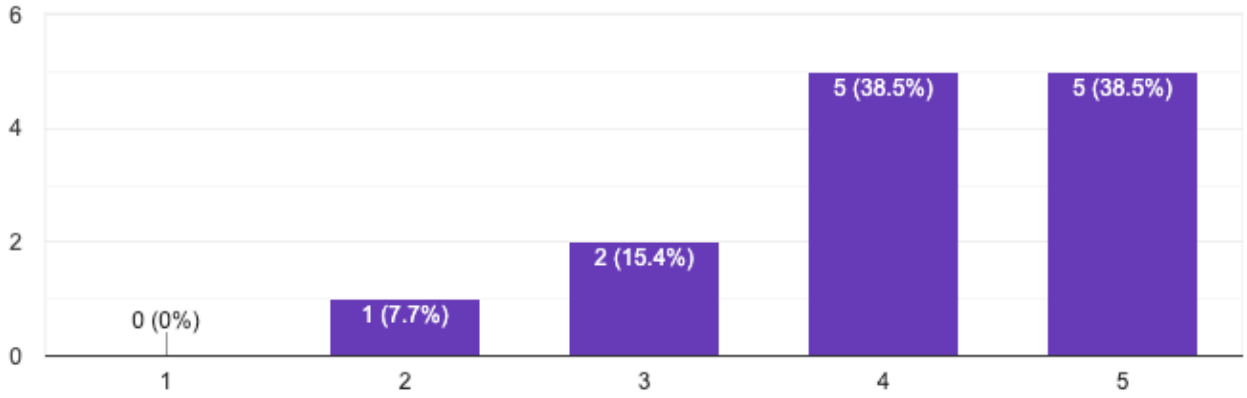
Answers:



3. How would you rate the user experience of the application? * (Single choice)

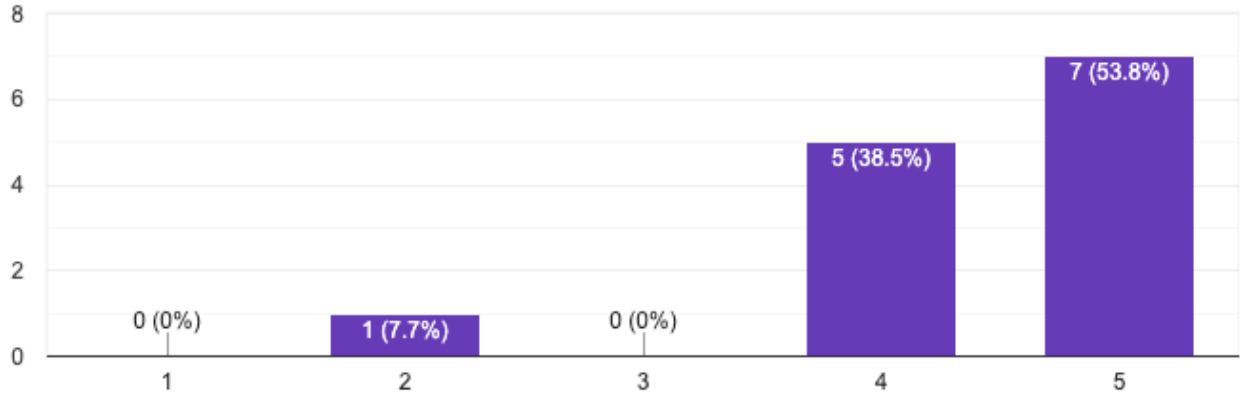


Answers:



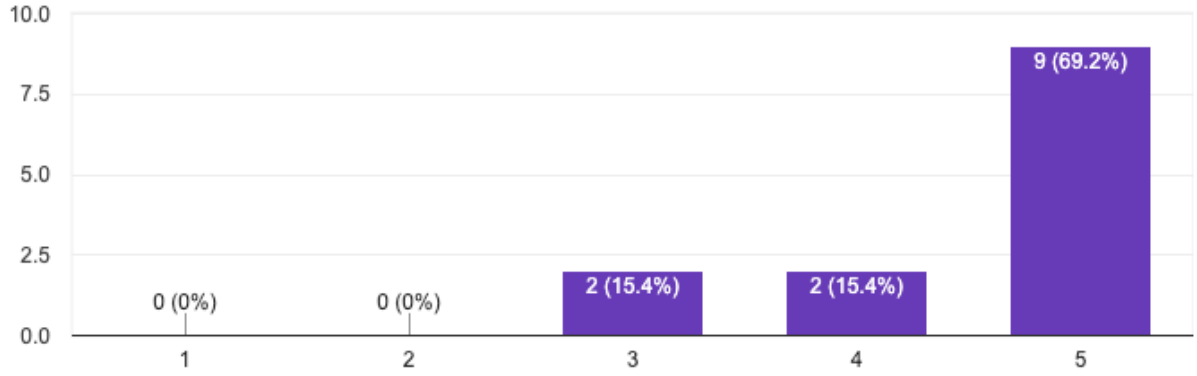
4. Does the new version affect the time you need to find the information you are interested in, in comparison to the old one? * (Single choice)

Answers:



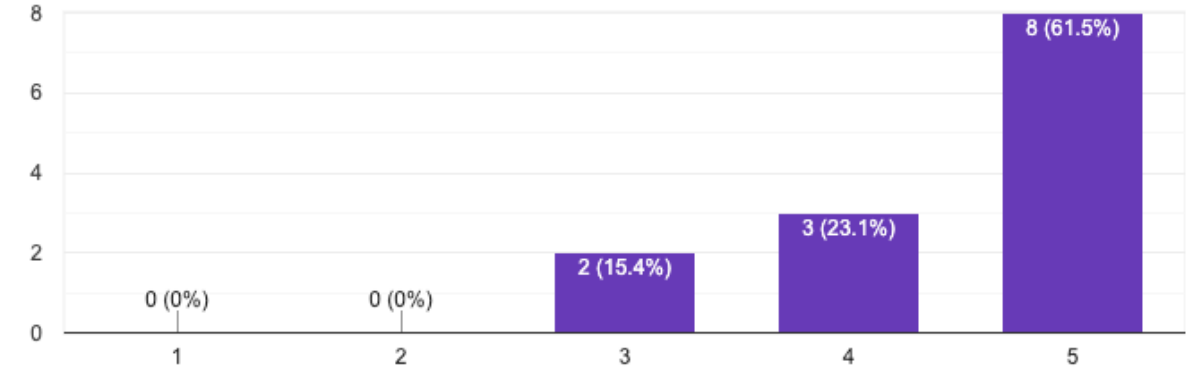
5. Does the new version affect the velocity of loading in comparison to the old one? * (Single choice)

Answers:

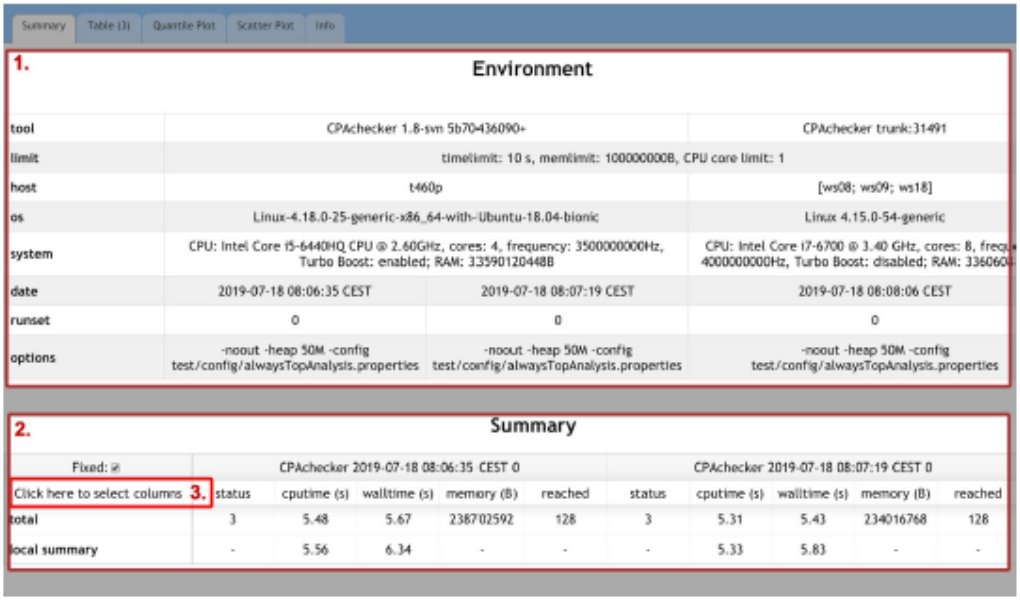


6. Does the new version affect the velocity of interacting with the table (click, sort, etc.) in comparison to the old one? * (Single choice)

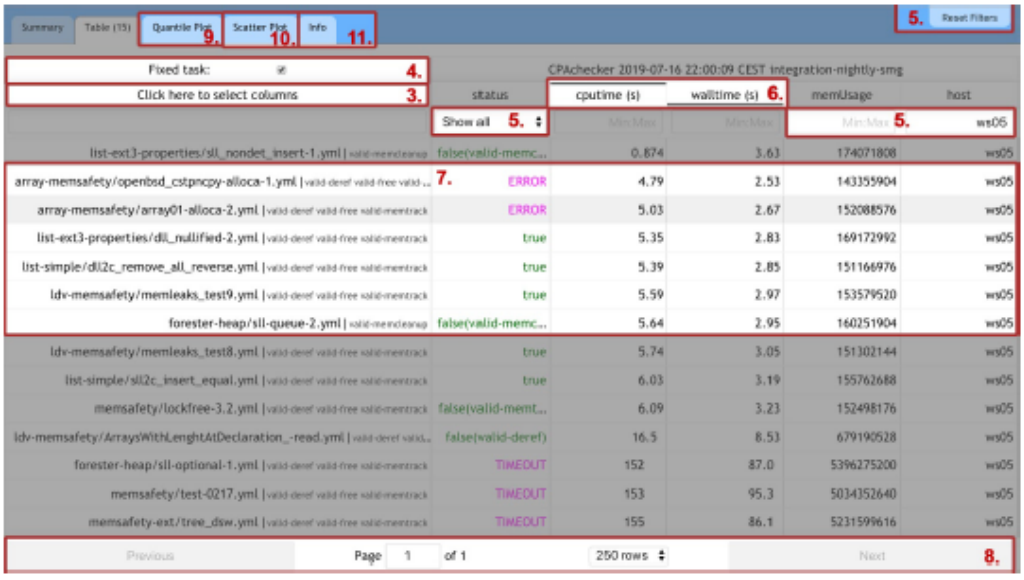
Answers:



Overview Features No. 1



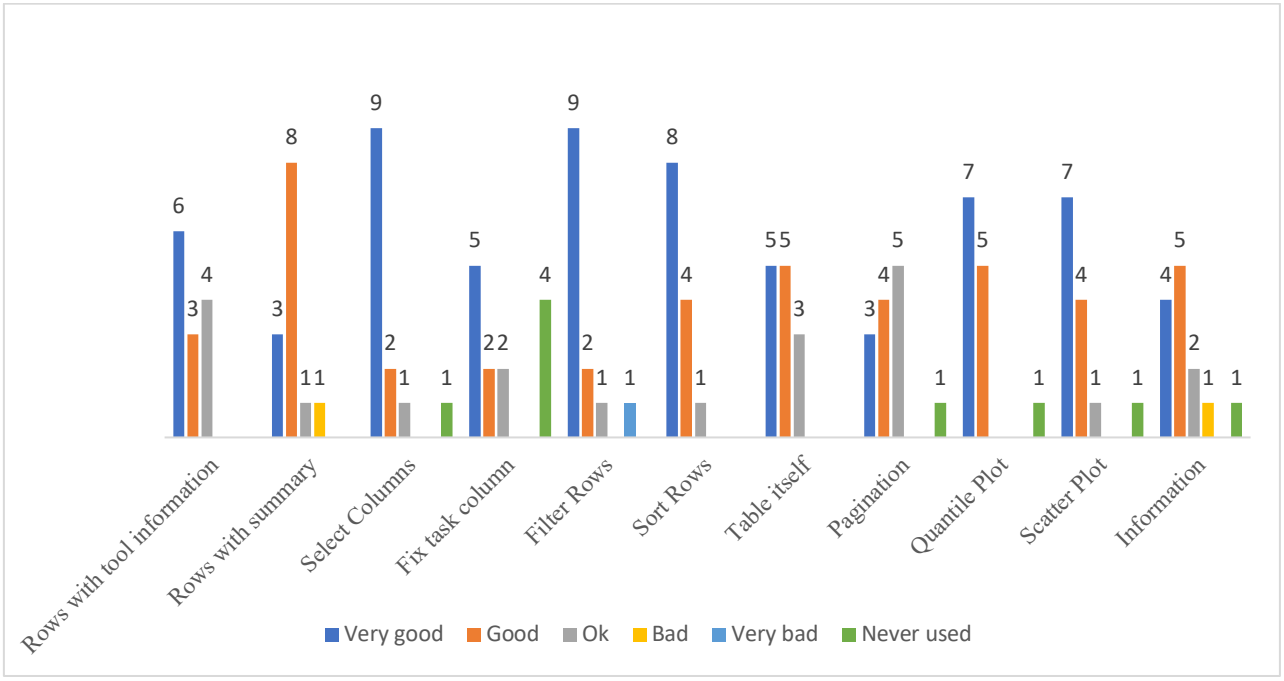
Overview Features No. 2



7. How well can the following functions be used? * (Single choice)

	Very good	Good	Ok	Bad	Very bad	Never used
1. Rows with tool information						
2. Rows with summary						
3. Select Columns						
4. Fix task column						
5. Filter Rows						
6. Sort rows						
7. Table itself						
8. Pagination						
9. Quantile Plot						
10. Scattem Plot						
11. Information						

Answers:

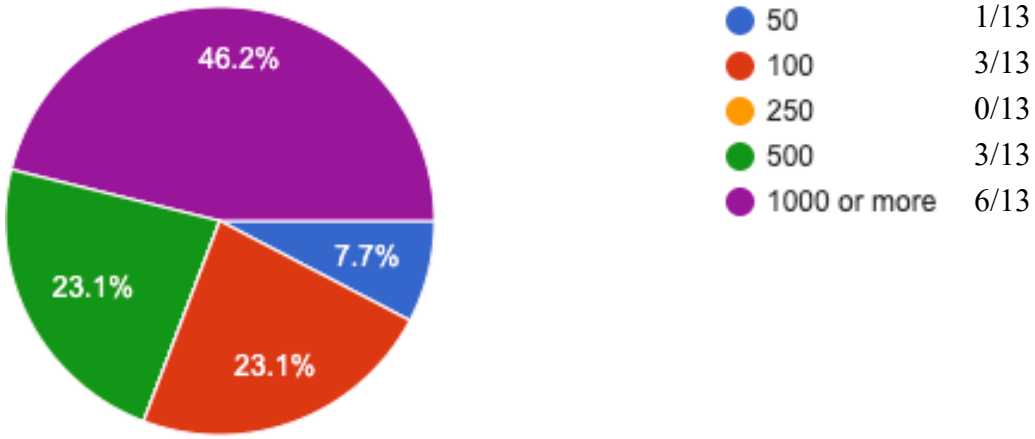


Settings

8. How many rows do you want to see on one page per default? * (Single choice)

- ◯ 50
- ◯ 100
- ◯ 250
- ◯ 500
- ◯ 1000 or more

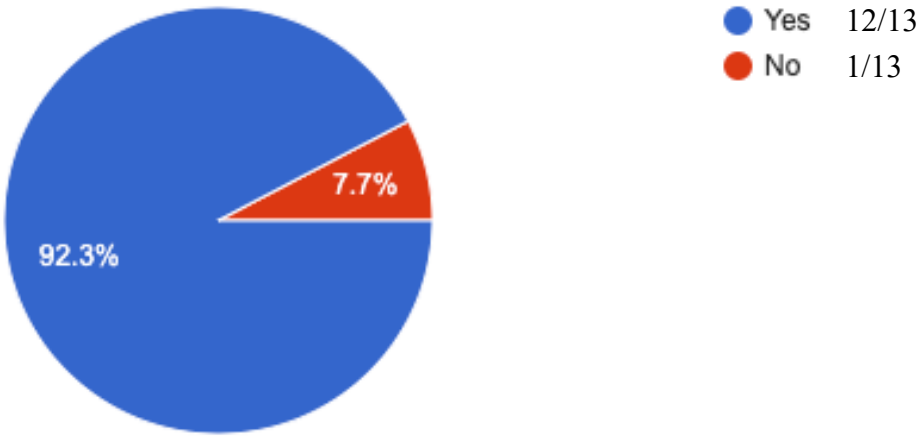
Answers:



9. Did you have a look at the info tab? * (Single choice)

- ◯ Yes *Skip to question 10.*
- ◯ No *Skip to question 12.*

Answers:



10. Did you miss any information (if yes, which one)? (Free text)

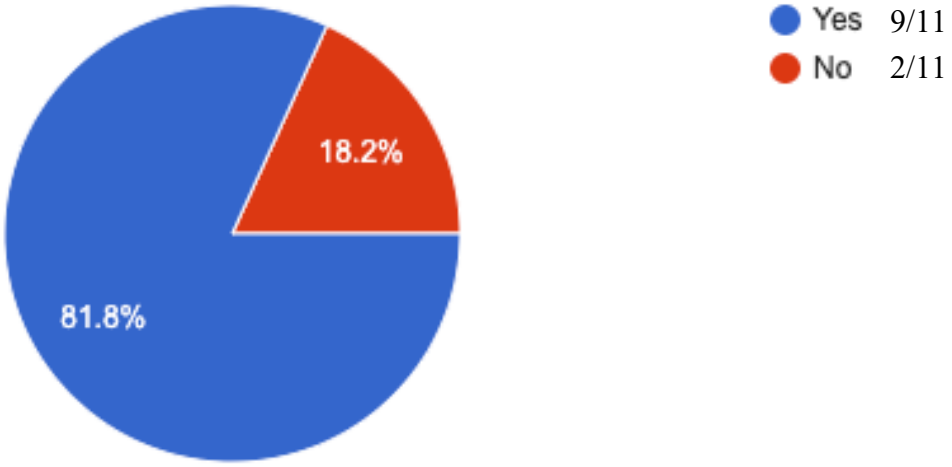
Answers:

- What does fixed task do?
- I miss the explanation for the fixed check box.
- I would add a column numbering the tasks
- no

11. Was the information helpful to find features and to understand how to use the application?

Yes
No

Answers:



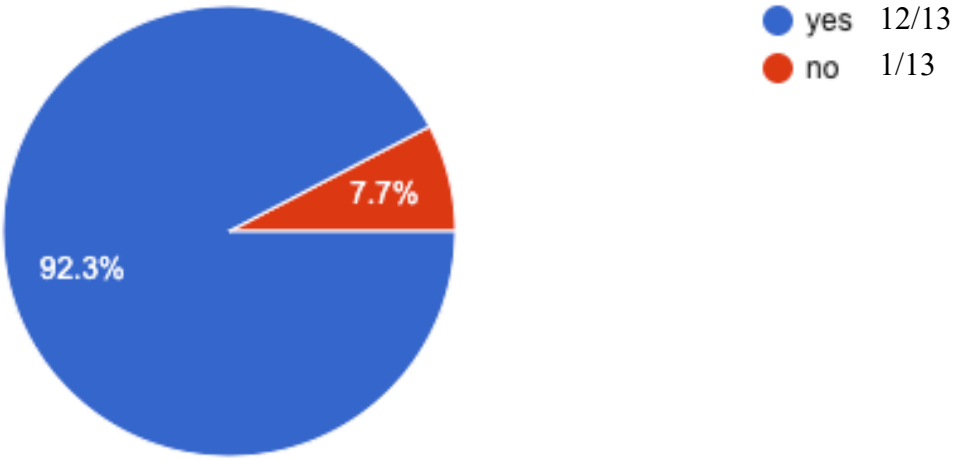
Final comments

In the end of this survey, you have some space to write down any hints, suggestions or ideas for possible next implementation steps.

12. Do you think the new version is an improvement in comparison to the old table application? * (Single choice)

- ◯ yes
- ◯ no
- ◯ Other: _____

Answers:



13. Please explain your answer from the question before (Free text)

Answers:

- Some minor points:
  - The number offset in case of float values is a bit strange.
  Maybe normalise the layout with a fixed precision: 10 -> 10.00
  - It would improve the readability if you thicken the lines between columns
  It is sometimes hard to tell to which a row belong
- much faster, modern interface
- For me the main improvement is velocity of loading the page.
- While loading works better, I heavily relied on the filtering function, especially the boolean combination of different stati. Moreover, it is now difficult to
- *HUGE* improvement, feels so much smoother to use
- Much better in every way!
- It is much faster and filtering in particular is helpful
- Filtering tasks and selecting columns to be displayed is a great addition. Having different tabs, including a documentation tab is also useful.
- Faster, more concise information

14. What is the biggest improvement and why? (Free text)

Answers:

- Pagination. I hope it will also improve the loading time for huge tables (like ECA-Rers in SV-Comp). In addition, I am a big fan of the summary tab!
- speed
- splitting parts (summary, table etc) into different tabs
- Velocity of loading the page
- The improved loading time.
- follows ux conventions, clear/friendly design
- performance and clarity
- see above
- See above
- faster loading time, better filtering, sorting
- Speed of plots

15. Do you miss any feature? (Free text)

Answers:

- Also minor: conversation from bytes to mega bytes or giga bytes
-
    - General small improvements: bold tab-titles. less blue space above the tab-title. the column separator color is light grey and is overseen very quickly. I missed that one table had one and the other one had three columns with benchmark results.
    - How do you sort tasks by alphabet? (just kidding, noone will ever sort filenames)
    - The new plots seem to be inline SVG. Is there an option to export this SVG directly to file? Would this be possible?
    - Could you rename the "info" tab into "about" and include the following info: version of BenchExec/TableGenerator.
- showing all results in single page (or paging percentage wise)
- I want more advanced filtering. I would like to have a summary that adapts to the filtering selection.
- (maybe comparing multiple result sets)
- a summary of the tool results instead of just correct/incorrect would be really helpful (perhaps cut-off when there are only 1% or so of results of some type)
- See above about numbering columns. I would also display memory in MB
- multi-select for filter on status column (e.g., show "wrong" and "ERROR")
- Plots can't be saved as pictures, when double-clicking column numbers it also selects whitespace after the numbers

16. Did you find any bugs? Please report them here. (Free text)

Answers:

- On Windows (using Chrome as Browser), once the window to the source code has opened, it can't be clicked away (except on switching the tabs)
-
    - sort by hostname does not work,
    - fast clicking on "fixed task" seems to freeze the UI,
    - mouse-over on column titles in summary-summary table does not show a click-curser, also the "click-here-to-select-columns" misses a click-curser.
    - the select-columns-overlay has movable (disabled) corners that are not needed and even overlay the close-button (Chromium browser)
- The spinner only changes the page after additionally pressing enter instead of changing imediately or after a short time limit.
- /
- no
- Some links are broken, e.g., https://www.cip.ifi.lmu.de/~wendler/programs/simple/pointerDereferenceWithNondetPointer_false-unreach-label.c when I click on that task

17. Is there a regression compared to the old table? What is it and why? (Free text)

Answers:

-
    - I found the green/red column toggle-buttons much better than the black/grey solution. the contrast is too low. this should be changed back.
    - in quantile plot: having the mouse over the legend text had put the corresponding quantile plot into foreground. this does no longer work.
    - the font in the plots is quite small. older people (like Philipp :-)) might need glasses.
    - the popup-info-hover-things in the plots seem to be left or right directed depending on the mouse position (left and right half). this was centerd before. Left/right alignment is also fine. However, if the window is too small on the right side, an additional movement border appears (Chromium browser).
- The filtering capabilities were more advanced in the old table and I used. Moreover, it is difficult to distinguish between different benchmarks, which makes comparison more difficult. This was better in the old design.
- for my taste, table font could be smaller to fit more content on screen
- /
- didnt see it, but perhaps when viewing a file -- in the sample tables you could not view files
- Don't recall.
- Some cells are too small to have their full content shown (e.g., "ERROR (interpolation failed)" in status column. It is also not possible to see the full value via hovering in a tooltip, only by resizing the whole column.

18. Space for your final comment: (Free text)

Answers:

- The new bench exec report is awesome and provide you with some useful features!
- Spam-bots will find the table. Please use a generic "benchexec@googlegroups.com" or reference the issuetracker, just not your personal email.
- I like the redesign and the new usability. Well done.
- Thanks for the work!
- Without boolean combinations of filtering the tables loose lots of value for me. Additionally, I do not like the design. Fonts seem to differ. One major problem is the difference in font size. Large numbers are hard to read and the memory columns dominate the table. Often, the fonts are larger than I would like to have them. The color gradient for the buttons in the plotting tabs puzzles me. The plots should use the same colors.
- Great improvement!! One thing, please don't use a gradient for the buttons. It is the only thing that diminishes the whole experience.
- Definitely a step in the right direction!
- :thumbsup: