Summary of algorithmic performances

Daniel Tircob * Bernhard Frick †

Faculty of Computer Science
University of Vienna

Index Terms: Human-centered computing—Visualization—Visualization techniques; Human-centered computing—Visualization—Visualization design and evaluation methods

1 Introduction

The lecture on Algorithms and Data Structures (ADS) includes an assignment in which each student has to implement a data structure for sorting a large amount of input data. There are multiple different data structures like hash tables or trees for the students to choose from for their implementation. Part of the assignment is to do a performance optimization. This task is successful if the student's implementation performs better than the supplied reference implementation. As an aide to reach this goal, the lecture provides a performance analysis tool that runs a number of tests, measures execution time and memory consumption and renders a time and memory consumption graph for each implemented function. Also contained in the graphs are the performance of the reference implementation and the performance of the other implementations.

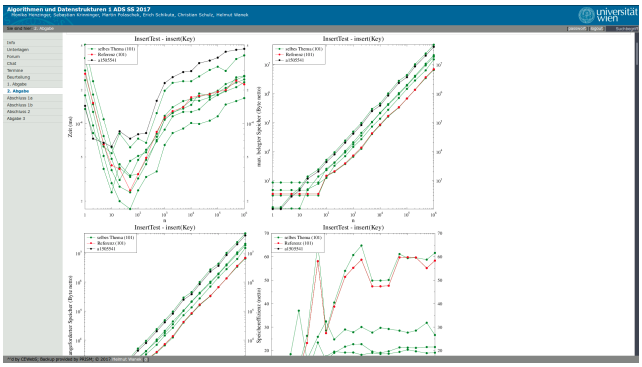


Figure 1: The current visualization on CEWebS

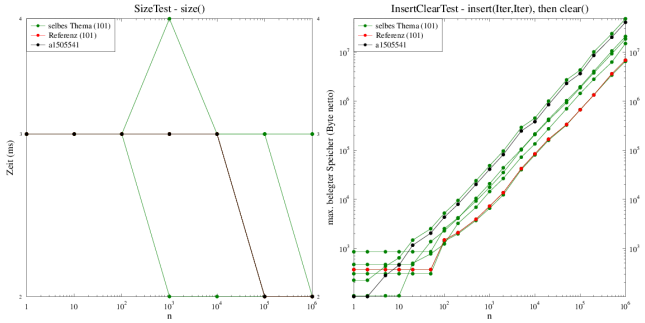
1.1 The Problem

The graphs are static server side generated images of graphs that are scaled according to the range of the input data. This results in seemingly giant differences when the data are in a small range, and significant differences disappear when the data are widely spread out.

An other issue with these graphs is that often different implementations are very similar to each other, which leads to a very cluttered mess of lines of the same color, making it extremely hard to identify which data points belong to which implementation.

The static images allow no interaction with the data, making it impossible to focus for example on one specific implementation graph or to change the scaling of a graph.

*e-mail: a01405569@unet.univie.ac.at
†e-mail: a01505541@unet.univie.ac.at



(a) Small differences seem large (b) Big differences seem insignificant

Figure 2: Scaling in the current visualization

Lecturers also use the same anonymized graphs as the students as a basis for the grading of the implementations. This means that finding outliers is only possible by clicking through all implementations until the matching implementation is found.

The generation of the images takes up to minutes, and every change from one of the students invalidates all graphs from all other students, which leads to them being regenerated on the next page refresh, which in turn again takes up to minutes.

There is a large set of graphs that are being produced and one really needs to focus on details to gain insight.

1.2 The Tasks

The main Objective is for students to identify bottlenecks in the performance of their implementation of the data structure in comparison to reference implementations and their fellow students using the performance evaluation system provided by the lecture.

The performance test is conducted by uploading the implementation to the CEWebS-platform, which executes a set of certain predefined operations like inserting or deleting different amounts of values in the data structure. Performance indicators are the measurements of the following criteria:

- Execution time (lower = better)
- Peak memory consumption (lower = better)

Based on the gained knowledge students are compelled to improve upon their implementations in order to qualify for bonus points counting towards the final grade.

1.3 The Users

We have identified the students doing the ADS lecture as the primary affected user group, as they rely on a useful visualization in order to qualify for a better grade.

The lecturers are using the same graphs to grade the implementations done by the students, with no enhancement in terms of non-anonymized data or similar features.

We determined the tutors to be the third affected user group, as they help the students find bugs in their code based on the unit test results.

1.4 The Data

The data given to us consists of one XML-File per user, identified by the student number. The root XML-Element is a test-tag with attributes to identify the student and theme. Inside that test, there are a number of test case-tags.

Figure 3: The XML-input-files

Each test case-tag describing the execution of one unit test method holds an attribute for the test type like an add-test, a search-test or a remove-test, and another attribute for one of the 19 different input sizes ranging from 1 to 1.000.000 data elements.

Inside the test case-tag contained, tags represent each measurement like the test input size, run time of the test or the peak memory consumption.

2 Related Work

As we already mentioned in the introduction part, the purpose of this project is to improve an already built system. Therefore, our main reference software is the one integrated in the CeWebs platform. See Figure 1.

After researching for little while, we also found the implementations of students from earlier years, that were debating the same topic.

Reading the papers of Elif Bilgin and Franz Brandl [Bil14] were somehow disappointing. Not only that they did not brought new ideas to the project, but the implementation was poorly made and the report were incomplete or not detailed enough, so you can understand exactly their point of view. It may be possible that the work of Cemil Seker and Haris Becic be a somehow better, but the only trace we could get was their website [Sek13], that does not look very promising.

Based on [Bil14] the only visualizations available are a line chart and a bar chart. The line chart shows the reference, the student and the average of one of the function that ”may” have been chosen. The bar chart should represent, conform the figures description, an overview of all the functions, where one of them is not behaving as it should. This charts are unfortunately not backed up by a concrete explanation, about how they really work. Same conclusion we can deduce also from Cemil Seker and Haris Becics website [Sek13].

A paper that made a little bit more sense was the one of Alexander Fomin and Hermann Hinterhauser [Fom17] written in 2016. They structured and completed their paper, used lots of screenshots as proof of the implementation and to guide the reader through their work.

If we would compare the types of implemented functions, with the other 2 papers, they also implemented line charts for the memory tests and placed all 3 graphs on the same page, similar to a dashboard, which makes the user get a better overview about how his implementation is doing.

One of the main reasons of their lack of results, may be because of the tools they used. All 3 projects were realized in java, with help of d3-library, [Fom17] also using Tableau for the Low-fidelity prototypes.

3 Our approach

Using the nested model for visualization design by Tamara Munzner, we started off analyzing the situation by creating personas and describing the goals of the users based on our own experiences when we took the lecture - this part is also clearly defined by the lecture and can be found in the ”Introduction” chapter.

3.1 Review of the current system

For the next step we met with the lecturers of the ADS lecture, Ass.-Prof. Mag. Dr. Martin Polaschek and Dipl.-Ing. Helmut Wanek, to discuss the existing data, the current design and expectations on a newly designed visualization.

What first catches the viewers attention about the old system is that graphs are often nearly indistinguishable from each other because they all use the same color. Having the ability to select different colors for single graphs and being able to select and highlight a graph would be a big help in that case.

What we determined unanimously was that scaling needs to be reworked in a way that somehow preserves relative distances between measures and not only shows the part of the scale where values reside in. we also talked about the possibility to change scaling between linear and and logarithmic.

Also a much requested feature by the lecturers is some kind of grading view that allows the display of names and student IDs in order to better identify which implementation belongs to which student, compare specific implementations and generally get more insight than the student have. Currently, lecturers are working with the same views that the student also see.

The last thing we talked about was the ”ratio”-measurement, which gives insight about the efficiency of the memory usage of the implementation. Currently there is no visualization of that measure.

We were told that an implementation in d3.js is not necessary, as the chance that this is really going to be used in CeWebS is fairly low and there would be nobody to maintain the needed infrastructure and visualization.

3.2 Low fidelity Prototyping

A basic decision we made based on the data and the current visualization design, is that it makes no sense to directly compare different tests for completely independent functions. Therefore, we think that the old approach of showing visualizations on a per-function-basis is a good one and we are building on that.

However it does make sense to also show all input sizes for one test in the same chart. That way, one can see whether an implementation scales linearly or if there are certain input sizes that cause massive time or memory consumption.

We came up with a range of various charts that suits our needs the most. From those, we chose only the ones that we found clean, simple and highly understandable. We then combined these graphics, so that the information that it reveals correlates with each other.

3.2.1 Bernhard Prototypes

One of the first things these prototypes address is sensible colors for the graphs. Choosing colors that also have a meaning and are easily distinguishable was an easy task. In order to support the reference graph even more, the background can in many cases be filled with a light red below or a light green above the reference, so that the performance of ones own implementation becomes clear faster.

Another thing that can be improved upon is the fact that in the old ADS-graphs there are indistinguishable lines for every implementation. This bloats the diagram and adds no real benefit. Instead,

it would be better to only show lines for average, best, worst implementations.

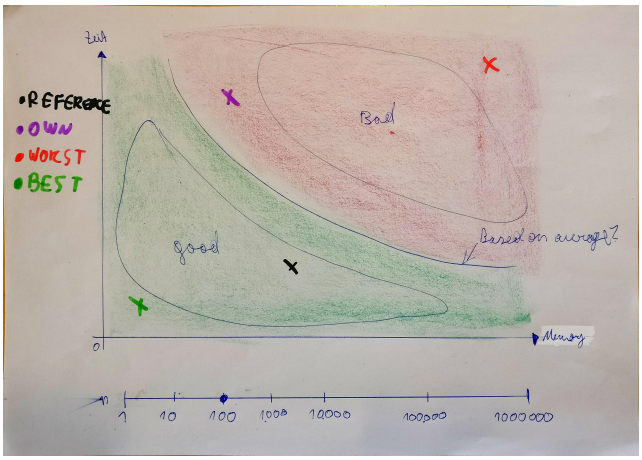


Figure 4: Prototype time memory scatter plot

Figure 4 shows a scatter plot for time- and memory-consumption for a given test-size n. Its design is based on the fact that very often, a reduction in time is only possible by using more memory and vice versa. Therefore, it could be possible to draw a logarithmic line that is some kind of benchmark. A good-performing implementation would be positioned somewhere near both axes and a bad implementation would be far away from both axes. In order to compare different n, a slider is introduced to select from a range of $[10^0, 10^6]$.

There is a clear disadvantage of this view: it is not possible to see the performance of an implementation at a glance, instead, one would have to scrub over the whole slider.

The advantage though is that similar implementations would show up near each other. Maybe it would even be possible to see some clustering happen, either within different implementations of a data structure or between different data structures.

What would certainly be hard is the calculation of the border between a good and a bad performing implementation. Maybe this can somehow be done based on the reference implementation.

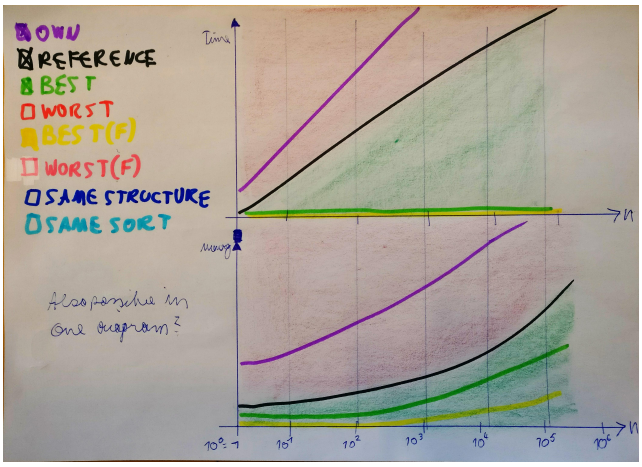


Figure 5: Prototype comparison with other implementations

In figure 5, time and memory are split up in two diagrams sharing the same x axis. It addresses the disadvantage of the previous chart by showing the test size on the bottom axis. The main objective of this view is to enable the user to choose a comparison on his own. With a list of check boxes to the left, one could select whether he wanted to see his own implementation, the reference, a good or a bad implementation, the average performance of all implementations, a fictional good or bad implementation where the performance is chosen for every n, or other implementations with the same structure or the same sort.

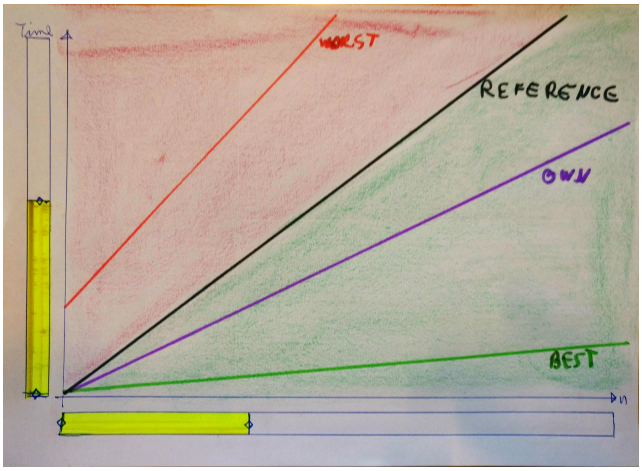


Figure 6: Prototype zoom for detail

Be it a line chart or a scatter plot, when having many different data points or implementations in the same view, it can get crowded and confusing very quickly. In order to overcome those difficulties and maximize insight for the viewer, figure 6 introduces two key features: first, brushing and linking is used to narrow down both axes so that only a specific area is displayed. In addition, a graph should have the possibility to be highlighted via hover and click. This enables the user to visually distinguish graphs that are very close together or similar in other ways.

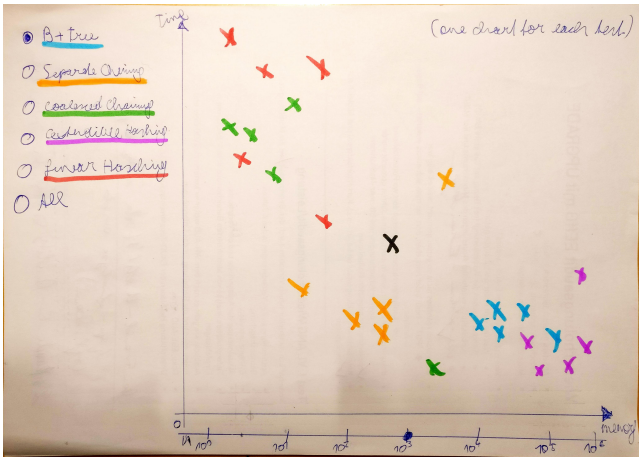


Figure 7: Prototype deviation from reference

Using the chart in figure 7, one should be able to see where students have the most problems and the largest successes implementing their data structures. It shows the most effective data structure over all and within data structures, and one can clearly see which implementations perform best. Again, a slider is used to scrub over n. Alternatively, this can also be implemented using two charts that share the same x axis (as described in the chart above).

In order to select the implementations to view, there could be a list of all implementations with a toggle button for each in addition

to a set of presets that allow to quickly select all implementations of one data structure.

There should be a reference implementation for each data structure. To make it stand out, one could use the same color but a different symbol.

3.2.2 Daniel's Prototypes

From early beginning, dashboards seemed a great idea. They allow the users to analyze and compare different parts of their program in the same time. An example could be; The project implementation of a student shows stunning results for the elapsed time for the "Insert function" for example, that inserts an array of values into the data structure. The memory it occupies on disk although, is extremely high. This way he gets the idea form the very beginning what could have gone wrong in his implementation.

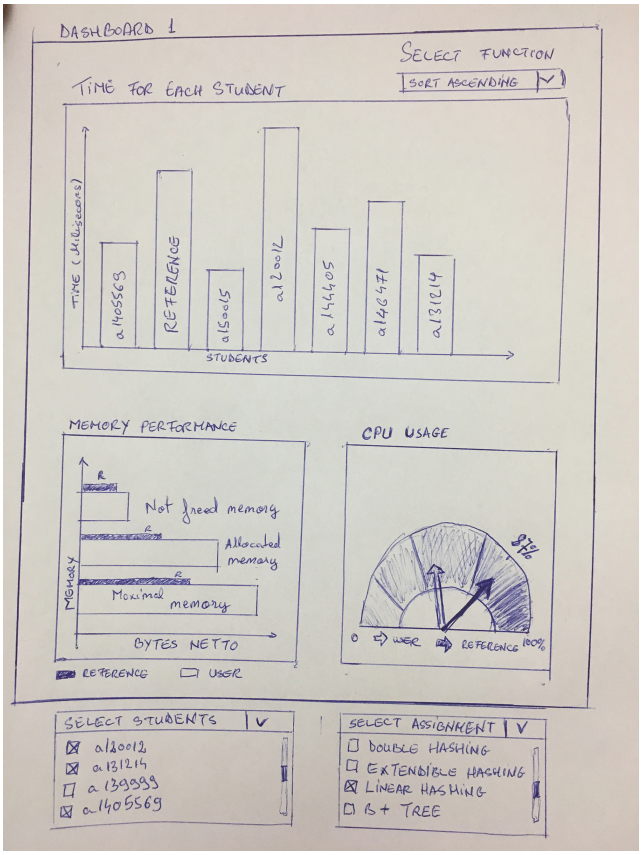


Figure 8: Prototype Dashboard 1

The above dashboard consists of three individual graphs.

The first view of the dashboard is representing the amount of time necessary for the sort function to finish. Here the "n" - number of objects that need to be sort - will be considered the highest. The bars that will always be present in this graph will be the student's bar, the reference bar and the best/worst time bar.

The second graph will give the students an overview about the memory performance. "Not freed memory", is the dynamic memory, like the heap, that in the lifetime of one test was demanded, but not freed. "Allocated memory" is the amount of memory the program is normally using. "Maximum allocated memory" is the maximum amount of memory that was needed during the test, the highest value of bytes needed to be stored during one trial.

The last graph is a feature that I think will be suitable for this assignment. It was not implemented or thought by anybody that we researched until now. We calculate the time, we calculate the memory but the amount of work that the CPU puts in is also important. This will be a good optimization counter. It will work basically as a heat map ranging from 0% to 100%.

An important technique that we would like to apply to all the graph will be tooltiping. In this way, by hovering over the time barchart, the user can see at exact millisecond, how fast his program/function was.

// image: Compare 1 apply function (sort function) // image: Compare 2add function (add into data structure)

Our second variation of the dashboard is somehow more colorful (The graphs will be updated in short time, possibly using a software tool like Tableau. The sketches are temporary.)

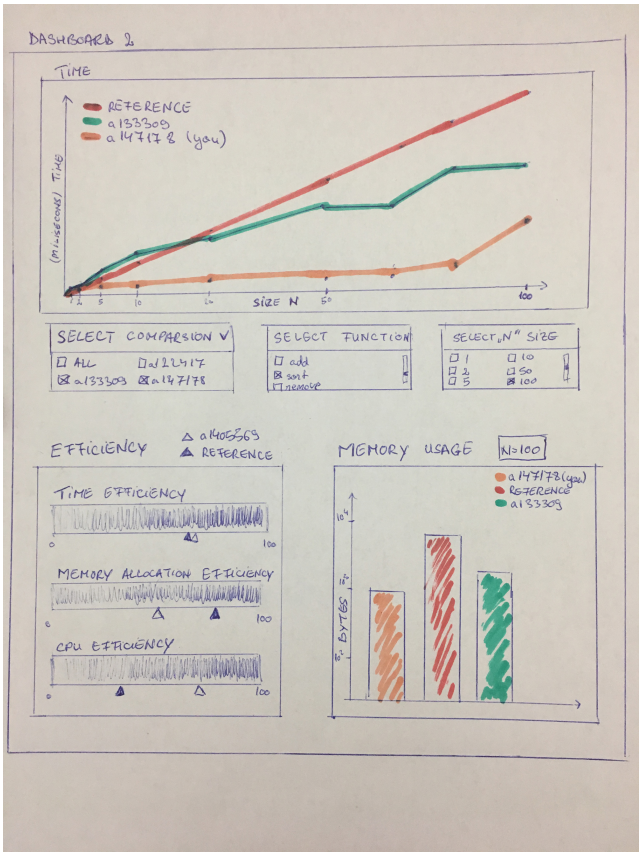


Figure 9: Prototype Dashboard 2

In comparison with the previous dashboard, we settled for a line chart for showing how much time a function needs. In addition, the filter for "n" will clarify how much of our data size we want to display. This way the user will be able to see how the program reacts at each step of data amount incrementation. Trying not to overload the visualization of the chart, filtering for some specific user fits perfect here. In case the person wants to see all of the other results, he can simply check the "all" box.

Colors also play an important role here. They will be set for each student, so in case of partial overlap, something will be still visible. Tooltipping will be an alternative in case the colors have very similar tones, like almost the same shade of green. This can easily happen, in case of huge amount of users.

Instead of only the efficiency of the processor, as in dashboard 1, we now present 3 types of efficiency, namely time, memory and CPU. These will all be represented through different heat-maps. Each efficiency type will have its own reference and all will range

between 0% and 100%. This will provide a nice overview of the entire program, because it aggregates all the important parts into one.

The last graph will manage the memory. It will have the same filter settings as set above. Having the ”n” specified is an advantage, because you can easily see how your program reacted at either small amount of data or a huge amount.

Because the legends of the graphs can get quite long, a scrolling window will be a wise option. This will increase space for all 3 graphs and will not clutter the software.

3.3 Implementation in Tableau

Based on the dimensions of the given data and the old ADS-graphs, is becomes clear that distinguishable colors are badly needed. While our color choices diverged a bit from what we had in mind with the prototype, we are still using colors to identify different implementations. Instead of many bluish and yellow near-invisible lines, we went for the color scheme that was presented in the VIS-lecture. It was a tedious task to set specific colors for all implementations, as colors had to be set for each data point of each test size of each functionality.

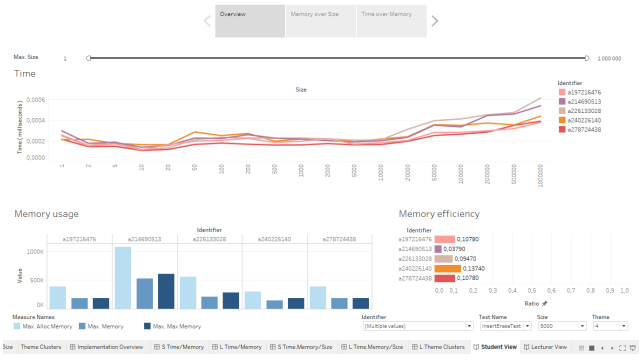


Figure 10: The student overview Dashboard

The student overview dashboard, as seen in figure 10, features graphs for execution time, memory consumption and even memory efficiency, that allow the comparison of ones own implementation with specifically selected implementations of your colleagues or a default selection of other implementations including the reference implementation.

Some things like strange positioning of labels might jump out in this view, which resulted from limitations in Tableau’s layouting possibilities, leading to big areas of unused space between charts when positioned correctly.

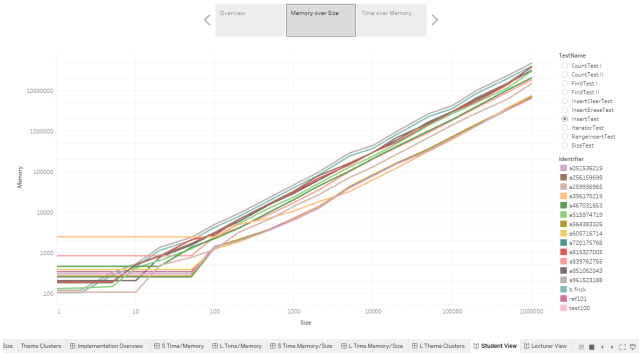


Figure 11: Memory usage analysis

Figure 11 shows the second page of the student dashboard, allowing in depth memory analysis. To the right, there are selects to choose which functionality to compare, and a list of other implementations of the same data structure.

The third view, as shown in figure 12, then combines the execution time and memory consumption chart into one scatter plot.

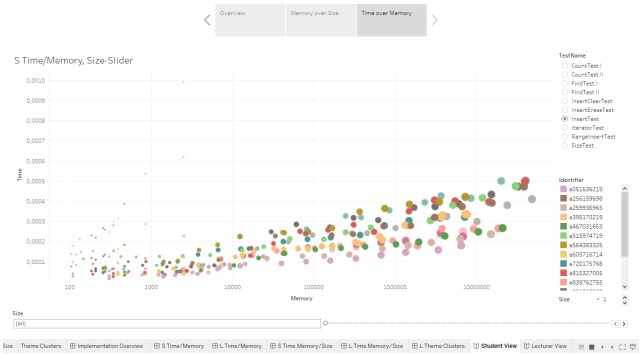


Figure 12: Time and Memory scatter plot

After digging through the performance data and getting a grasp of how the individual functions might perform in a general sense, we found out that some of the prototype graphs might not work the way we figured. One such example is the time memory scatter plot of the student view (shown in figure 12), where the basic idea was to see kind of a trade-off between these two measurements. This turned out to not really be the case, as most implementations kind of performed in the same neighborhood of time and memory consumption. There were some exceptions of course, but those did not exhibit an inverted time-memory consumption but were clearly far away from everything else.

The second part of our design is a lecturer view, which offers even deeper insight and detail. For example, lecturers are enabled to see which implementation belongs to which student by the indication of a name and the student id.

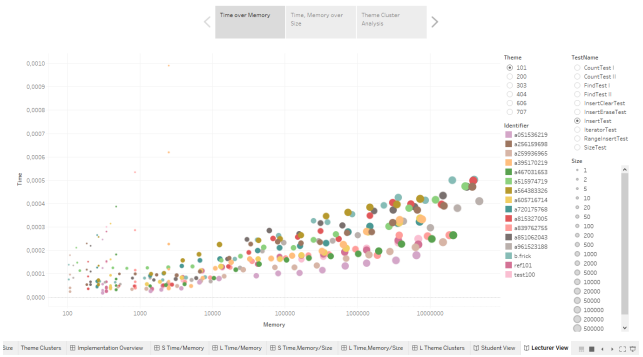


Figure 13: Time and Memory scatter plot

The lecturer view (shown in figure 13 starts off with a time memory scatter plot, which we thought results in the best overview over all implementations. There are controls to switch between different themes and functions. We deliberately decided to implement these switches as radio buttons, and hereby disallowing the comparison of different functions of the implementation. Comparing two different parts of code performance wise makes no sense. For example, inserting 1 million values into the structure takes a big amount of memory and time, and the code needed to do this is in most implementations over hundred lines long. Looking up one value in the data structure

on the other hand is a task of 3 lines and is executed extremely quickly.

The second page of the lecturer dashboard (figure 14) is made specifically for analysis of performance trade offs, as such behavior can hardly be spotted in a scatter plot. Note that here, the upper execution time graph uses linear scaling and the lower memory consumption graph uses logarithmic scaling. We found out that this seems to fit the available data best.

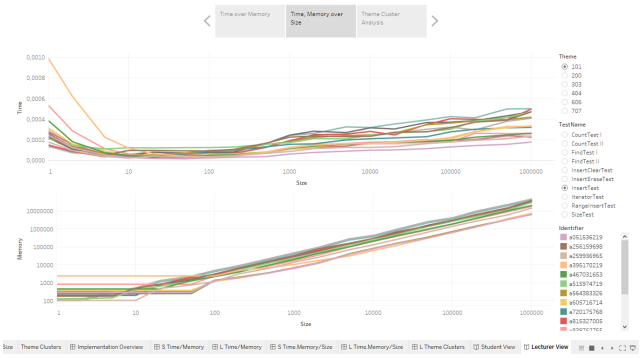


Figure 14: Time and Memory usage analysis

Finally, on the last page of the lecturer dashboard, as shown in figure 15, we placed a completely new graph that gives insight about how different data structures or themes perform. Per default, all implementations of all themes in all sizes are shown. The only constraint is the test (as mentioned before). Colors are used to differentiate between themes this time. Still, it is possible to select one student via the tool tip that appears when clicking on one data point. It also becomes visible which theme was implemented most often and that within some themes, deviations are larger or smaller. We see that the best and worst performing implementations overall are of the same theme. This makes for a nice overview over all data structures, and might even be valuable when teaching the students about the data structures and their performances.

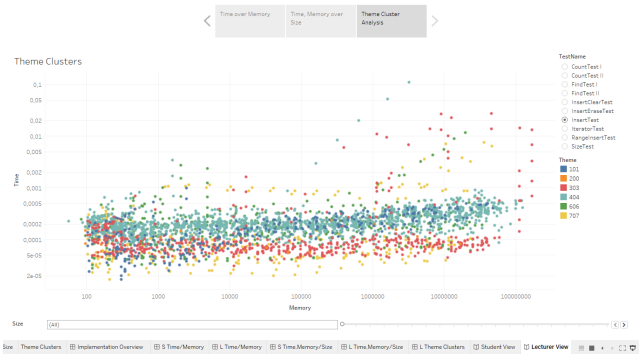


Figure 15: Theme cluster analysis

on the bottom we included a slider that allows the selection of different test sizes. This slider only allows the selection of one size or all sizes. Using check boxes for sizes would enable a comparison of the performance using 10.000 elements vs. the performance of using 1.000.000 elements, but we left this option out because Tableau would not allow the combination of the check boxes and the legend for the size of the dots, resulting in the display of two ”size”-lists, one for check boxes, and one for the legend.

Some of the aforementioned graphs are placed on different dashboards, because putting them on the same dashboard would require them being half-sized or even smaller, making them useless again because the many data points would be too tightly packed and nearly invisible.

This leads to these graphs not being linked. We did research about linking between different story pages and our findings can be read in ”Challenges” below.

4 Implementation Details

The performance data we received consists of one XML-file per student, containing an array of tests that are run, each with different input data sizes, which in turn contain the measurements for time and memory (see chapter ”Introduction/The Data” above).

4.1 Data preparation

We used the standard Scala XML Library and its query functionality in a custom Scala script to extract data from the XML-files in order to convert them to a CSV-file, which is the most convenient format to import into Tableau.

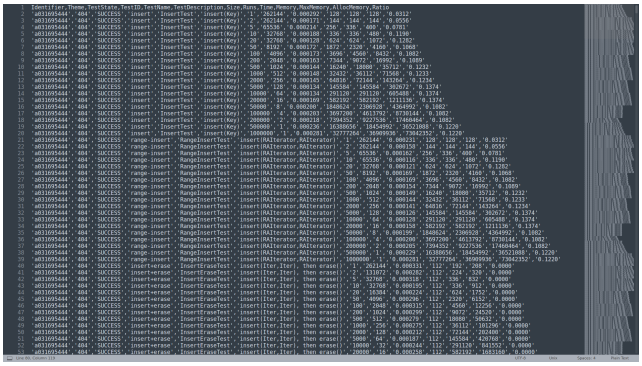


Figure 16: The converted CSV-files

4.2 Visualization

After deciding that this project is not meant to replace the current visualization found on the CEWebS-platform, we opted for Tableau as our visualization tool, instead of implementing a web-based visualization tool with d3.js. Tableau is a WYSIWYG-Editor for visualizations that offers an easy to use interface that allows to quickly create beautiful and interactive visualizations via drag-and-drop interactions and also offers deeper control via mathematical operations that can be applied to the data and a basic set of user interface elements for somewhat custom controls. This also remedies the fact that with a web based implementation we would also have to design the complete user interface.

4.3 Challenges

The hardest problem we encountered is creating custom controls in Tableau. Sliders, radio buttons etc. are not fully customizable to the extent we wish they were, and there is no option to combine them with a legend for example. One instance of such is the legend for the dot-size that indicates the test size. Showing a legend while also allowing filter-like functionality on the test size, would force us to display the list of all sizes two times, each with a different purpose. In this case we decided to stick with a slider, not allowing the selection of different sizes.

It is also not possible to apply a filter to multiple pages of a Tableau story. Upon further research, we found that this is an open feature request at Tableau, but Tableau decided that pages are not meant to be used in such a way. This might indicate that we used Tableau the wrong way and our design indeed needs some rework.

Furthermore, we did not yet find out how to do something like colors based on RegEx-matching or similar. This would enable us to

automatically highlight the reference implementation graphs as their IDs begin with ”ref” instead of ”a”. We tried different approaches using grouped data and other things, but none lead to the wanted result. Custom coloring seems to be an extremely tedious task.

Most of our problems we are sure could be mitigated by re-implementing the design using d3.js, as it offers nearly endless customization possibilities, but this task is out of scope and left to the reader or a follow-up project on the matter.

5 Results

5.1 Scenario: Student

Yvonne is a CS-Student in the second semester, attending the algorithms and data structures course. She wants to receive all the possible bonus points for the best-performing implementation of the data structure.

As our visualization design is not meant to replace the current design, we are not going to go discuss the submission and performance testing process.

Yvonne is presented with the dashboard showed in Figure 10. This contains measures of the execution time and memory consumption. The overview also provides selects to change between tests and different input sizes.

She represents the student with ID number a240226140 therefore, the orange line in the chart, that she could either manually search for it or just type in the search box. Beside herself, she also selected other peers, by going to the bottom of the view and clicking the ”Identifire” tab.

Figure 17: Filters of the student dashboard

Because all the persons that she selected had somehow same results as she had, the chart is now cluttered and the lines are not clearly distinguishable. So she decides to reduce the data size, seeing only the results for data between 200 and 500000. She simply drags the slider form both sides, until the desired value is met, or she can insert the minimum value to the left input size of the slider and maximum value to the right of it.

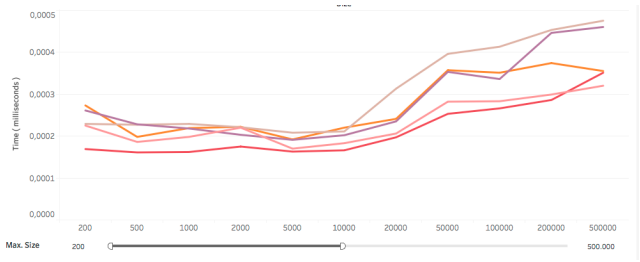


Figure 18: Input values in slider

This did not help too much. It is still not clear if she is better than the purple line (for example). So she wants to see a more detailed result. In this case, by tooltiping on the lines, a box with precise informations about is popping up. Now it is much clear for her, at what size she got that time result.

Now she would like also to see how the memory usage in relation with the size is. By clicking her own ID in the Identifire legend on the right side, everything about that specific user will be highlighted, leaving the others in the background. This way she can concentrate more on her results and also keep an eye on the others or the reference graph.

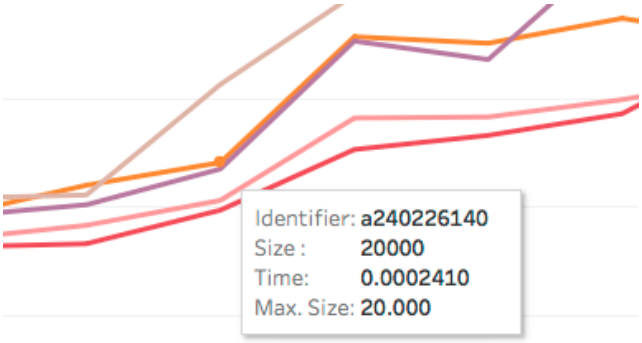


Figure 19: Tooltip

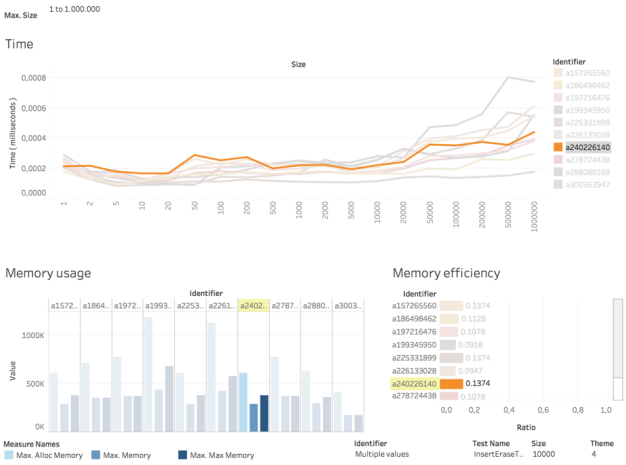


Figure 20: Highlighting

Advanced charts then provide deeper insight: the time/memory-chart displays each test as a point on a scatterplot. Tests of the same implementation share the same color, the input size is denoted by the size of the dot. Selecting a dot highlights all data points that belong to the same implementation. Using the legend to the right, multiple implementations can be highlighted to faciliate a direct comparison.

In this view, it becomes clear quite quickly, that the own (yellow) implementation is performing worse than all others in the lower input sizes. By clicking the dot, they all get highlighted. Hovering the dot then reveals the input size that results in such overblown time consumption.

5.2 Scenario: Lecturer

The main goal for a lecturer in order to grade the implementations is to easily spot deviations from the reference implementation, be it significantly better or worse.

Different themes are individually graded this time he reviews the implementations of theme 101, which is the B+Tree data structure. In the first step, the lecturer view is opened with the Time over Memory scatter plot as the default screen.

Using the view in figure 21, he gets an overview of all implementations of the B+Tree data structure, separated by the functionality tested (like inserting or deleting data in the structure). Colors indicate different implementations and the size of the dot indicates the test size, which is in the range of $10^0, 10^6$. For each theme, the reference implementation is included (we could not manage to

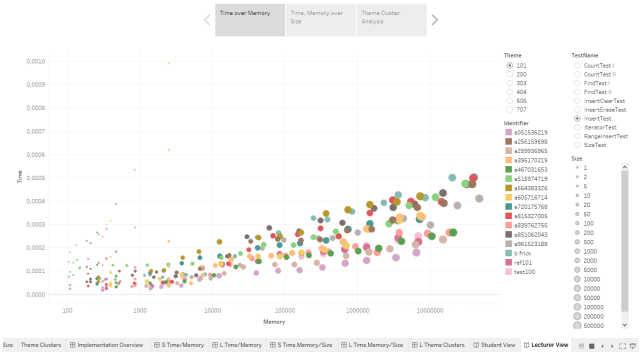


Figure 21: The default screen of the lecturer view

display it in a way that stands out from the others using tableau). Here, one can already see outliers for the first time: there are some yellow dots high up on the left side.

To further inspect this anomaly, the lecturer then clicks the yellow dot, which brings up a tool tip showing information like the test size and the implementing student.

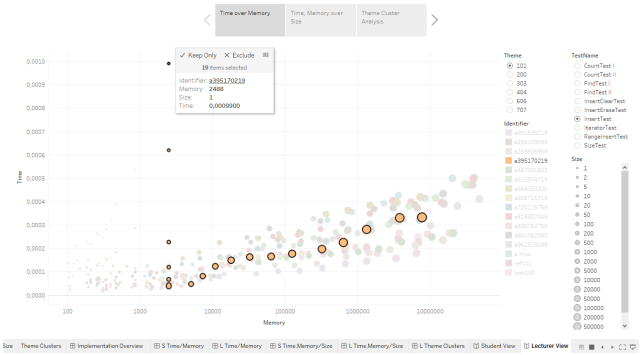


Figure 22: Selecting all data points of one implementation

Upon clicking the identifier in figure 22, all test sizes belonging to the implementation are highlighted.

Now it is clearly visible, that the implementation has some serious issues with input sizes below 10.000 elements. Above that, it performs just like all other implementations.

For direct comparison to the reference implementation, we now also select the reference in the legend to the right, which is shown in figure 23.

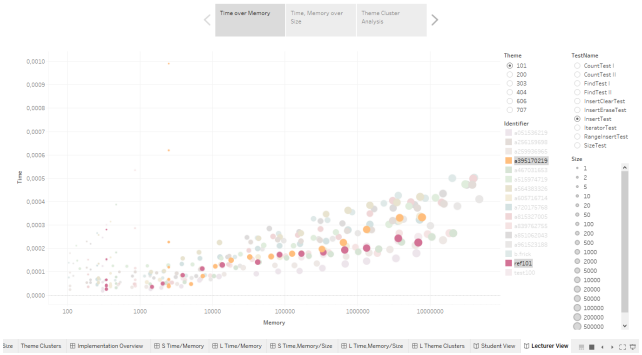


Figure 23: Direct comparison to the reference implementation

We see now that the graph of the reference implementation kind of looks the same, but with significantly lower execution time apparently, they used a similar approach, thus the resembling graphs, but the student implementation of the insert function is off by one magnitude in the lower test sizes.

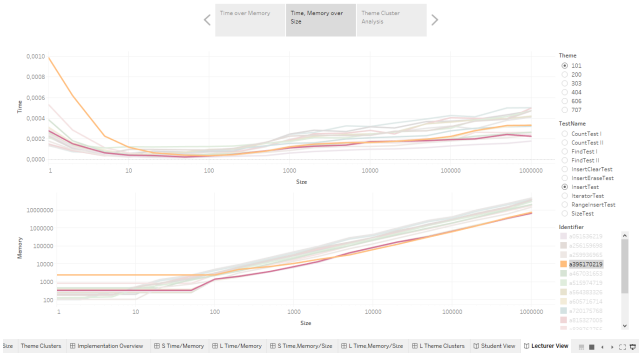


Figure 24: Clearly visible trade off in separated view

Consulting the second visualization in figure 24 showing two separate diagrams, we can identify that there seems to be a trade off in memory usage the implementation performs badly in the lower ranges, while becoming on par with or in some parts even outperforming the reference. Execution time however does not see any improvement in the higher ranges.

Now it is up to the lecturer which grade to give generally, these data structures are meant to handle humongous amounts of data, which means that the seemingly catastrophic performance in the lower ranges might be an acceptable trade off, especially when considering the logarithmic scales used ins these visualizations.

5.3 Performance

Overall, we are quite happy with the capability of our design, especially proven through the fact that we could easily find many anomalies while playing around and not even specifically searching for performance issues. We both implemented such a data structure on our own and our implementations were included in the data set too. Despite the fact that the data we got was anonymized, it was still really interesting to dig through.

The design is still rough on the edges, but we think that we achieved our goal and the tool works fairly well for a high fidelity prototype.

5.4 Feedback

We showed our prototype to about 5 fellow students that all have already passed the ADS lecture. Our designated task for them was to see if there is an issue with a given implementation (we provided the student id), and if yes, what the reason might be.

In our analysis, 4 out of 5 people liked the idea of overview. They found it interesting that the size is bounded together with how the memory was affected at that point in time. Mainly they liked interacting with the dashboard, seeing what happens if they choose more people, if they click the specific person and they found tooltiping very useful because they could see everything.

The other person thought that 3 graphs is too crowded (taking in consideration that the experiment was done on a 13” Macbook)

Something that was harder to cope with all of them was understanding the functions themselve. This has but not everything to do with the design, but with the task itself. It took us also a lot of time understanding the difference between all the memory types and which was which in the data set.

6 Discussion

What our design does especially good is showing the user at a glance where anomalies reside. This is made even easier by offering details via interaction and the ability to directly compare two specific implementations without displaying all other graphs. We addressed most of the concerns raised by Ass.-Prof. Mag. Dr. Martin Polaschek and Dipl.-Ing. Helmut Wanek like distinguishable colors, better scaling and a grading view and many issues with the implementations in our data jumped out even when we were just toying around and not really specifically searching for problems.

All in all we think that our design really did improve upon the current design and is most likely the best implementation compared to previous projects done on the same topic (see chapter "Related Work").

Still, there is room for improvement: some diagrams could be combined and linked in one big dashboard, enabling even more interaction between them. With some of the graphs though, this is not an easy task, as they would soon become unreadable again because they are so small. This is definitely an area with potential.

Tableau also makes it hard to implement custom controls, like combining check boxes and legends or sliders, which are nearly impossible to style apart from default functionality like displaying the 0 value or not.

Cross platform usability is bad with Tableau, and this is especially important as the home of c++, the language the students use for their implementation, is on Unix and there is no Unix version of Tableau. A web application would be able to address that. And finally, implementing our design in JavaScript using d3.js would allow a much more fine grained customization of controls, legends and other UI components and even makes integration with CEWebS possible.

What we learned during this project, is that in order to do a visualization task right, there is a tremendous amount of work to do and there is no real possibility to somehow make it less work, like with every task where users are involved. We know that we could have done better if we invested more of our time. Also it is a pity that our design was not meant to replace the current graphs, but we understand that this is out of the scope of the visualization lecture.

7 Work log

Both of us did half of the user testing with the prototypes, showing it to our friends.

We both did some minor corrections on the Tableau prototype, concerning layouting and things that were rushed in Milestone 3.

Writing the paper was split on a per chapter basis, resulting in the following work share:

- Introduction: Bernhard
  - The Problem
  - The Tasks
  - The Users
  - The Data
- Related Work: Daniel
- Our Approach
  - Review of the current System: Bernhard
  - Low Fidelity Prototyping: Daniel Bernhard
  - Implementation in Tableau: Bernhard
- Implementation Details: Bernhard
  - Data Preparation
  - Visualization
  - Challenges
- Results
  - Scenario Student: Daniel
  - Scenario Lecturer: Bernhard
  - Performance: Bernhard
  - Feedback: Daniel
- Discussion: Bernhard

Acknowledgments

The authors wish to thank Ass.-Prof. Mag. Dr. Martin Polaschek and Dipl.-Ing. Helmut Wanek, the lecturers of the Algorithms and data structures Course, for supporting us with both performance data and feedback on our designs.

References

[Sek13] Becic Haris Seker Cemil. *Algorithm-based Performance Visualization*. June 2013. URL: http://cemil-seker.wixsite.com/vischproposal.

[Bil14] Brandl Franz Bilgin Elif. *Visualisierung fffdfffdfffd Project fffdfffdfffd Report*. June 2014. URL: vda.univie.ac.at/Teaching/Vis/14s/project_finalReports/team11.pdf.

[Fom17] Hinterhauser Hermann Fomin Alexander. *Summary of Algorithmic Performances*. Jan. 2017. URL: vda.univie.ac.at/Teaching/Vis/16w/Final_Report/02.pdf.