Internship report

Paolo Marzolo

May 2021

1 Introduction

I was tasked with creating software (based on a web app) that would help evaluate the correctness of a user's mental model of a concept after the user had read the explanations generated by this tool, which explains the logics followed by a credit approval system, developed as part of a recent paper [1]. The activity was estimated to take around 120 hours, and was to be completed in May 2021. What follows is a report on the problems we faced during development, and what we did to solve them. I will divide them into two main sections: Conceptual problems, and Technical (or implementative problems).

The need for this software, as mentioned in the initial proposal, is obvious:

> Let's suppose that any algorithm for the generation of user-centred explanations produces a so-called Explanatory Space (ES), that is the space of all the possible explanations reachable by any user through that algorithm.
>
> Evaluating the goodness of such an algorithm is the same of evaluating the quality of its ES.

2 Conceptual problems

2.1 Sizes and Brains

The size of the entire Explanatory Space is unmanageable by any single human being: because it is generated by crawling thousands of webpages, it contains every subject mentioned in extreme detail, and is not only incommunicable to the user, but also incomprehensible in its entirety at any one time. Because of this, we could not use the entire ES as "grounds" for testing, because, first of all, it would be unreasonable to expect that a user explored all of it, and secondly, it would contain sections that are only tangentially relevant to the explanation itself.

2.1.1 Solution and advantages

We decided to only use what had been rendered in the webpage as our source of truth: this accomplished two things.

1. It allowed us to keep the size manageable by the user: we only used what the user viewed, so it is reasonable to assume the user read the pages that were loaded.

2. It kept our Portion of Explanatory Space (PoES) mapped one-to-one with what was seen by the user, not only for what concerns the concept shown, but also in what detail.

2.2 Mapping with explanatory space

The nature of the Explanatory Space cannot be completely reduced to a simpler medium (if we assumed so, that would require a second experiment to evaluate the correctness of such a hypothesis).

2.2.1 Solution

We decided to maintain the original "shape" of the ES (a graph), and give the user a way to interact with it

2.3 Graphs are complex and unbounded

We suspected giving the user complete freedom to create any graph that represents his or her own mind map would result in maps that, although may be reducible to one another by a human, would be completely unrelated when considered by a similarity algorithm. To avoid another layer of explainability, we needed the user to submit a possible solution that was at the very least comparable to the PoES that was explored.

2.3.1 Solution

To achieve this, we kept the possible nodes bounded: the only nodes that could be added were the ones in the original graph, and context was given for how and where they were mentioned. So the user received a very specific portion of the Portion of Explanatory Space they had explored, one at a time, and connected the nodes in the correct spot.

2.4 Context and bounded options make for an obvious problem

Clearly, when given complete context and fully bounded options, the problem was greatly reduced in complexity, so much so that any alert user would make no mistake even if the ES was not shown at all.

2.4.1 Solution

To combat this, we added a random subset of other nodes to the pool the user could choose from; these nodes were pulled from the pool of nodes the user had analyzed in its exploration, so they were contextually relevant.

3 Technical problems

In this section, I will focus mainly on usability concerns, as performance has only one great problem: complex graphs are tough to layout, handle, and render, especially in JavaScript; this is even worse when adding complex nodes with internal logic (such as the collapse nodes we used).

3.1 Too many edges can't be followed

When creating the solution, as soon as edges (gaps in the text) increased beyond one or two, it became complex to follow where a specific edge was going to, and which handle it came from.

3.1.1 Solution

We used two different approaches for this issue:

1. We implemented a selection system for gaps, which allowed the user to click a gap to select it; once a gap is selected, a red outline is shown around it, and the edge that represents it becomes animated.

2. We included the content of the node that was connected to the handle directly inside the collapsable node: this separated the concerns in two distinct sections, as the edge + node group was only relevant for changing the edge or eliminating the node, while reading the text was delegated to the collapsable node itself.

3.2 Nodes get overwhelming quickly

In the view mode, because of how the ES is constructed, nodes multiply quickly, and soon become hard to follow (and hard to render!).

3.2.1 Solution

To avoid this, once the graph reaches a certain threshold (left to the administrator of the test) the nodes that only had an output role are hidden (as the information they contain is already in the preceding collapse node. An additional solution (but more technically challenging) would be to collapse the questions inside the concept as well.

3.3 Navigation helps too much

We quickly realized that, while completing the various questions, having the context for the later questions gave part of the answer to the previous ones.

3.3.1 Solution

We sidestepped this by simply removing the "back" button inbetween questions.

3.4 Integration with Vue and the previous application

The tool built in [1] is built on another tool, so both implementations are in Vue. This was scarcely compatible with our use-case

3.4.1 Solution

While this suited the previous use, we decided to switch to React, for an easier implementation of our reactive graph.

4 Conclusion

This summarizes the main challenges we faced, from both a conceptual and a technical point of view. More information can be found in the commit history of the project, located here. As one could expect, a lot of time was dedicated to technical implementation that were not even mentioned here (for example, the graph layouting algorithm).

We believe the tool we created represents a useful starting point for evaluating the usefulness of the aforementioned explanatory tool.

References

[1] Francesco Sovrano and Fabio Vitali. From philosophy to interfaces: an explanatory method and a tool based on achinstein's theory of explanation. In *Proceedings of the 26th International Conference on Intelligent User Interfaces*, 2021.