Identifying K-12 Students' Approaches to Using Worked Examples for Epistemic Programming

Sven Hüsing*
sven.huesing@uni-paderborn.de
Paderborn University
Paderborn, Germany

Sören Sparmann*
soeren.sparmann@uni-paderborn.de
Paderborn University
Paderborn, Germany

Carsten Schulte
carsten.schulte@uni-paderborn.de
Paderborn University
Paderborn, Germany

Mario Bolte
mario.bolte@uni-paderborn.de
Paderborn University
Paderborn, Germany

ABSTRACT

Programming, when practised in an epistemic sense, can be used as a means of gaining insight into areas of personal interest that extend beyond the programming domain itself, e.g. by analysing data or running a simulation. However, novice programmers need support in conducting such personal epistemic programming projects. This paper reports on an eye-tracking study, aimed at identifying different approaches to using worked examples in this context. The study, conducted with 26 participants from K-12 using SMI REDn eye trackers, aims to uncover different types of (heuristic and presumably epistemic) approaches. In this context, we present an approach to clustering and aggregating scan paths through using dynamic time warping. The results show five different clusters with different behaviours: While some participants had several linear reading phases in the worked example, others focused more on parts that were necessary for their individual efforts. This could be an indication of epistemic programming.

CCS CONCEPTS

• **Social and professional topics** → **K-12 education**; • **Applied computing** → **Interactive learning environments**.

KEYWORDS

Epistemic Programming, Worked Examples, Eye Tracking, ScanPath, Dynamic Time Warping, K-12, Programming Education

ACM Reference Format:
Sven Hüsing, Sören Sparmann, Carsten Schulte, and Mario Bolte. 2024. Identifying K-12 Students' Approaches to Using Worked Examples for Epistemic Programming. In *2024 Symposium on Eye Tracking Research and Applications (ETRA '24), June 04–07, 2024, Glasgow, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649902.3655094

*Both authors contributed equally to this research.

ETRA '24, June 04–07, 2024, Glasgow, United Kingdom

ACM ISBN 979-8-4007-0607-3/24/06
https://doi.org/10.1145/3649902.3655094

1 INTRODUCTION

In many disciplines, programming is used to gain knowledge and insight [Odden et al. 2022; Wilensky et al. 2014]. Here, the programming process results in an intertwined interaction between cognition and programming [Hüsing et al. 2023] in the context of which learners can explore areas of personal interest and find answers to individually relevant questions. This perspective on programming is denoted as epistemic programming [Hüsing et al. 2023] and is currently being investigated as an approach for programming novices and K-12 students [Hüsing et al. 2024].

However, as programming can be difficult for novices to learn [Robins et al. 2003], students need to be supported in their epistemic programming endeavours. A recent study using retrospective think-aloud interviews has shown that worked examples can be used as such guidance to support novice programmers in engaging in epistemic programming [Hüsing et al. 2024]. A worked example denotes a worked out (programming) solution for a project similar to the one at hand [Atkinson et al. 2000; Mulder et al. 2014; Sweller and Cooper 1985; Sweller et al. 1998]. Students can use it as a guidance or as a basis for their own processes by using, adapting or expanding parts of it (also see the Use-Modify-Create approach [Lee et al. 2011]). In particular, so-called heuristic worked examples [Reiss and Renkl 2002] seem to be particularly appropriate for epistemic programming as they are supposed to convey the heuristic process behind the choice of specific implemented actions [Mulder et al. 2014]. While there is already research on using worked examples for teaching programming and in programming activities [Muldner et al. 2023; Rahman and du Boulay 2010], we want to investigate whether they can be beneficial for epistemic programming projects and how they might be used in this regard. Therefore, we want to identify and analyse different approaches to using worked examples in this context. In other words, we want to reveal different types of (heuristic and presumably epistemic) approaches and want to learn more about how learners interact with worked examples, and which approaches are beneficial for engaging in an epistemic programming practice. Since analysing eye-tracking data might potentially be a suitable approach to investigate cognitive processes within programming endeavours [Busjahn et al. 2014; Just and Carpenter 1976], we developed an eye-tracking based approach for examining K-12 students' processes of utilising worked examples within epistemic programming projects. In this context, we analyse, in particular, the extent to which students are guided

by the structure of the worked example, the extent to which they consider certain sections in it, and how the process of reading the worked example is intertwined with the programming process.

Following this approach, we report on a first analysis of students' behaviour in their epistemic programming endeavours in this paper. Therefore, we conducted an eye tracking experiment in which the participants were given a heuristic worked example to solve a programming task. The task description was designed to encourage an epistemic programming practice. We used eye tracking to determine which parts of the notebook participants were looking at at any given time. To identify students' approaches, we applied Dynamic Time Warping (DTW) [Kumar et al. 2019; Sakoe and Chiba 1978] to cluster and aggregate students' scan paths.

2 BACKGROUND

In this section, we provide a background to the concept of epistemic programming processes and the use of (heuristic) worked examples in this context, before elaborating on analysing programming processes through eye-tracking.

2.1 Worked Examples to Support Epistemic Programming Processes

As previously mentioned, programming offers various possibilities beyond software development. One such possibility relates to gaining insights through programming that extend beyond the programming domain itself, a practice already employed in various disciplines [Odden et al. 2022; Wilensky et al. 2014]. Epistemic programming [Hüsing et al. 2023] aims at addressing this perspective for novice programmers in particular, in order to teach programming as a means of exploring personal interests. However, since learning to program might potentially be difficult [Robins et al. 2003], beginner programmers need to be supported in their intertwined cognitive and programming processes. Here, worked examples [Atkinson et al. 2000] appear to be a promising approach [Hüsing et al. 2024], serving as a template for the programming process. Using worked examples is commonly known within programming education research [Muldner et al. 2023]. More generally, [Mulder et al. 2014] describe studying worked examples as being beneficial regarding algorithmic solution processes [Atkinson et al. 2000; Sweller et al. 2011]. Regarding inquiry-based learning, which is aspired in the epistemic programming approach, the authors describe that so-called heuristic worked examples [Reiss and Renkl 2002] might be particularly applicable in order to exemplify such inquiry-driven problem-solving processes and offer a guidance for students [Mulder et al. 2014]. Regarding epistemic programming endeavours, we want to identify a suitable design for worked examples, so that 1) programming novices are provided with actual code, they can use, adapt and expand [Lee et al. 2011] and 2) people, inexperienced with strategies regarding discipline-specific investigation processes are provided with suitable heuristic approaches [Mulder et al. 2014]. In order to meet these requirements, we assume that a computational essay [DiSessa 2000; Odden et al. 2022; Perez and Granger 2015; Wolfram 2017] could be used as a worked example in order to "help students think and work in a particular area, modified appropriately by the student for her expressive purposes" [DiSessa 2000, p. 185]. For identifying programming novices' approaches of utilising such worked examples, analysing eye-tracking data appears to be a suitable method for having a look into the intertwined cognitive and programming processes [Busjahn et al. 2014; Just and Carpenter 1976].

2.2 Analysing Programming Processes using Eye Tracking

The analysis of eye movement can provide insights into various aspects of programming cognition, such as code comprehension, debugging, problem-solving strategies, cognitive load, or programming expertise [Bednarik and Tukiainen 2006; Obaidellah et al. 2019]. A key concept in studying gaze behaviour involves analysing scan-paths, which depict the sequence of eye fixations on the stimulus: "Sequential analysis of scan-paths is required to understand the flow of visual attention on a task" [Goldberg and Helfman 2010, p.228]. However, analysing individual scan-paths is only of limited use for detecting shared patterns that can be generalised. In order to derive common patterns between subjects, eye-tracking research often involves clustering and aggregating multiple scan paths. As [Goldberg and Helfman 2010, p.227] state, "methods to identify similar scan-paths and aggregate multiple scan-paths have been elusive". In this section, we describe several commonly used methods in this field. Many of these methods applied are based on the concept of comparing string-based AOI (Area of Interest) sequences [Goldberg and Helfman 2010]. A common metric for calculating the distance between AOI sequences is the *Levenshtein* distance [Brandt and Stark 1997; Levenshtein et al. 1966]. First, the AOIs are mapped to an alphabet of single characters. Each AOI sequence is then converted to a string by concatenating the sequence of characters. The Levenshtein distance between two strings is then calculated as the minimum number of substitutions, insertions and/or deletions required to transform one string into the other [Goldberg and Helfman 2010]. A more intricate approach to compare string-based AOI sequences is the use of sequence alignment techniques such as pairwise or multiple sequence alignment (MSA) [Burch et al. 2018; Khedher et al. 2018]. The aim of these techniques is to find common sub-sequences in the scan paths. This "is a challenging task since they are typically not equally temporally long, do not consist of the same number of fixations, or do not lead along similar stimulus regions" [Burch et al. 2018, p.1].

Hidden Markov Models can be fitted to multiple AOI sequences to develop probability distributions for the AOI transitions [Goldberg and Helfman 2010]. However, these models do not take into account the history of transitions and therefore are of limited use in analysing scanning strategies among multiple participants.

Other approaches rely on spatial and temporal metrics such as fixation duration [Eraslan et al. 2016], saccade length and saccade direction to group subjects [Kumar et al. 2018].

A more recent approach to deal with the temporal differences in participants' scan paths is to use Dynamic Time Warping (DTW) [Kasprowski and Harezlak 2019; Kumar et al. 2019; Sakoe and Chiba 1978]. DTW is an algorithm for measuring similarity between two temporal sequences, which may vary in length and speed, by calculating an optimal match between the two sequences, similar to the Levensthein distance. For example, Kasprowski & Harezlak [Kasprowski and Harezlak 2019] have used DTW to generate a

warped time distance chart for comparing multiple scan paths based on spatial properties. In the following, we describe a new approach, using DTW on dwell times within intervals, to aggregate and visualise the visual flow of multiple participants in order to ultimately derive students' approaches.

3 METHODOLOGY

3.1 Experiment Design

In order to identify and analyse different approaches regarding the use of worked examples in the context of epistemic programming, we conducted an eye-tracking experiment with K-12 students in which the participants had to solve a programming task supported by a worked example. The task was to choose their favourite summer holiday resort from three pre-selected cities. In order to come to a decision, they were given different weather data from the three locations (temperature, humidity, particulate matter and air pressure), which they could analyse within a Jupyter notebook using Python. In accordance with the epistemic programming approach, the participants were free to carry out their analysis in terms of choosing any data set, computing any visualisations and measures, as well as using any kind of filters on the data with regard to their individual preferences. However, they were asked to justify their decision based on their analysis and their personal preferences.

The 26 participants enlisted for our experiment were drawn from three K-12 computer science courses from grade 10. The participants had already limited experience with the programming language Python, as they had already programmed in Python in class beforehand.

3.2 Material

The participants could conduct their analysis and write down their decision in a prepared Jupyter notebook, containing information about the task and the data sets provided as well as a map showing the three pre-selected cities (see the right Jupyter notebook in Figure 1b). Within the Jupyter notebook, they could write code into code cells and were given a Markdown cell at the end of the Jupyter notebook for documenting the most important findings and the decision.

In order to support them in their intertwined cognitive and programming-process, we prepared a worked example in form of a computational essay on an analysis of the weather in London from the previous year. The worked example contained information on visualising data in a scatter plot and a box plot, filtering data (by month and by time), and comparing data in a combined visualisation (multiple scatter plots and multiple box plots). For each aspect, a text cell with explanations of possible use cases and usage, a code cell and a cell with the output from the code were given. These cells were then used as Areas of Interests (AOIs) for the eye-tracking experiment (see Figure 1a). The participants could explore the worked example independently, orient themselves regarding the methods (visualisations, filters) applied there and use parts of the given code with regard to their individual endeavours. During the experiment, the screen of the participants was divided into two halves: While the worked example was shown on one half of the screen, the other half contained the prepared Jupyter notebook for the participants to conduct the data analysis and capture their findings (see Figure 1b).

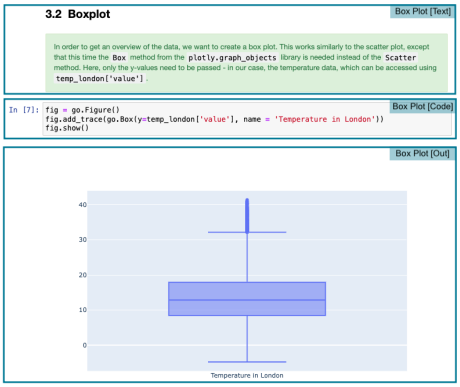
3.3 Data Collection & Analysis

We collected gaze data from the participants throughout the experiment using JuGaze [Sparmann et al. 2023]. JuGaze is a tool for conducting eye-tracking experiments in Jupyter notebooks, where each cell is treated as an AOI. During the experiment, JuGaze captured the position and size of each cell in the notebooks along with relevant metadata (e.g. cell type, cell selected). For the eye-tracking we used SMI REDn eye tracking devices set to 60 Hz. We extracted fixations from the raw gaze data using the I-HMM algorithm [Salvucci and Goldberg 2000]. By mapping fixations to cell geometries, we were then able to determine when and for how long each cell was fixated. In order to compare individual approaches, we divided each trial into intervals of 60 seconds to compare global and local patterns in the approaches. By resampling the data into fixed time intervals, we deliberately abstracted away from local patterns, prioritising an overarching perspective that aligns with our primary focus. We calculated the AOI dwell times for each interval in each trial, resulting in a time series $x = (x_1, \ldots, x_n)$, where n is the number of intervals in the respective trial. Each x_i is an m-dimensional vector

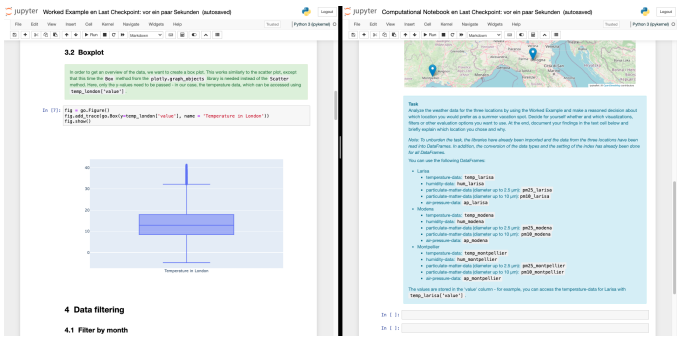
$$x_i = \begin{pmatrix} x_{i1} \\ \ldots \\ x_{im} \end{pmatrix}$$

where m is the number of AOIs and x_{ij} is the dwell time on the j-th AOI in the i-th interval. When represented in a $m \times n$ matrix, the time series can be visualised as a heat-map, with one axis representing time as a unit of intervals and the other axis representing the AOIs. An example of such a heat-map can be seen in Figure 3. We then applied Dynamic Time Warping (DTW) [Sakoe and Chiba 1978] to calculate the pairwise distances between the participants' time series in order to compare their approaches. We used cosine as the base metric for DTW as it is better suited for dealing with the sparse vectors derived by the large number of AOIs. By using DTW we were able to compare and cluster time series of different lengths while preserving information about the overall order of cells visited. In addition, it allows multiple trials to be aggregated using DTW Barycenter Averaging (DBA) [Petitjean et al. 2011] to visualise clusters and generalise the respective approaches. However, due to a bug in the used version of JuGaze, in some cases, the output of the user-generated cells was not captured by the tool. This only affected cells created during the experiment and only occurred in a small proportion (2 of 10) of the trials we examined. As a result, some of the fixations on the visualisations created by the participants were incorrectly mapped to the corresponding source code AOI. However, we did not omit the data for these AOIs, as these errors appeared to have only a marginal effect on cluster composition and may still be helpful in understanding what the participants were doing.

In order to evaluate and compare the quality of the participants' notebooks, the number of (combined) visualisations and filters created was determined. Furthermore, we checked whether the reasoning behind the decision was based on the participant's results,



(a) Section of the worked example showing the AOIs (turquoise) assigned to the Jupyter notebook cells.



(b) Arrangement of the students' screen during the experiment: one half shows the worked example (left), the other half shows the Jupyter notebook for the students' analysis and documentation (right).

Figure 1: Experiment setup and stimulus.

indicating that the participant had gained insights through the programming process.

4 RESULTS

As described in section 3.3, we calculated pairwise DTW distances for all trials based on the dwell times within each interval. Figure 2 shows the resulting hierarchical clusters based on the DTW distances using average linkage. We applied a distance threshold t to form flat clusters. The threshold value determines the number of clusters: a lower value produces more clusters and vice versa. We chose $t := 30$, which resulted in 5 clusters, each containing from 2 to 13 participants. This was the maximum number of clusters that could practically be visualised and compared side-by-side, allowing us to differentiate between the different approaches as best as possible within this limitation.

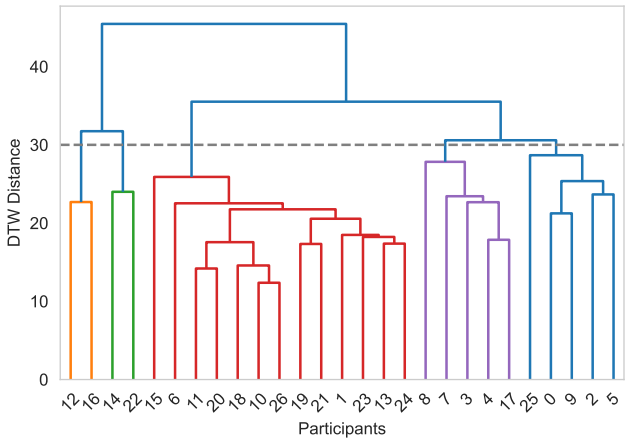


Figure 2: Agglomerative clustering of participants based on their dynamic time warping distances. The clusters are coloured based on the threshold $t := 30$.

Figure 3 shows the aggregated approaches for each of the 5 clusters as a heat-map based on the DTW Barycenter Averaging. The colour of the cells represents the calculated dwell time for the corresponding interval. The duration of the aggregated clusters varies from 31 minutes (cluster 2) to 53 minutes (cluster 4). In addition, the results of the assessment of the participants' notebooks are shown in Table 1.

5 INTERPRETATION & DISCUSSION

Based on the heat map in Figure 3, we describe and interpret this visualisation to give a general overview of the aggregated clusters. Additionally, we refer to the evaluation of the participants' notebooks, for which corresponding performance metrics were documented in Table 1.

The two participants from cluster 1 seemed to have a reading phase in the worked example, starting at the beginning of their process and lasting approx. 10 minutes. After this, they again looked into the parts of the worked example for creating scatter and box plots as well as for filtering the data by month. At that point of time, there are also longer dwell times on the cells with custom code. In the very end (after around 28 minutes), the participants apparently also have written down text, while looking into their outputs. As it can be seen in the participants' Jupyter notebooks, one of them could not arrive at a final decision and only created one visualisation. The other participant was able to apply a filter (regarding specific months) and created a combined visualisation.

In the heat-map for cluster 2 ($n = 2$), three phases of scrolling through the worked example can be recognised (at the beginning, after approx. 10 minutes and after approx. 25 minutes). Additionally, the dwell times on custom code increased throughout the process. However, there were lower dwell times in the custom text cells. Participants in cluster 2 apparently spent a lot of time reading the task or looking at the map and little time on working on their code or reading in the worked example. This also becomes apparent from looking at their produced notebooks as one of them was not capable of producing any visualisations and the other one did not compare the data in a combined visualisation and did not use any filters.

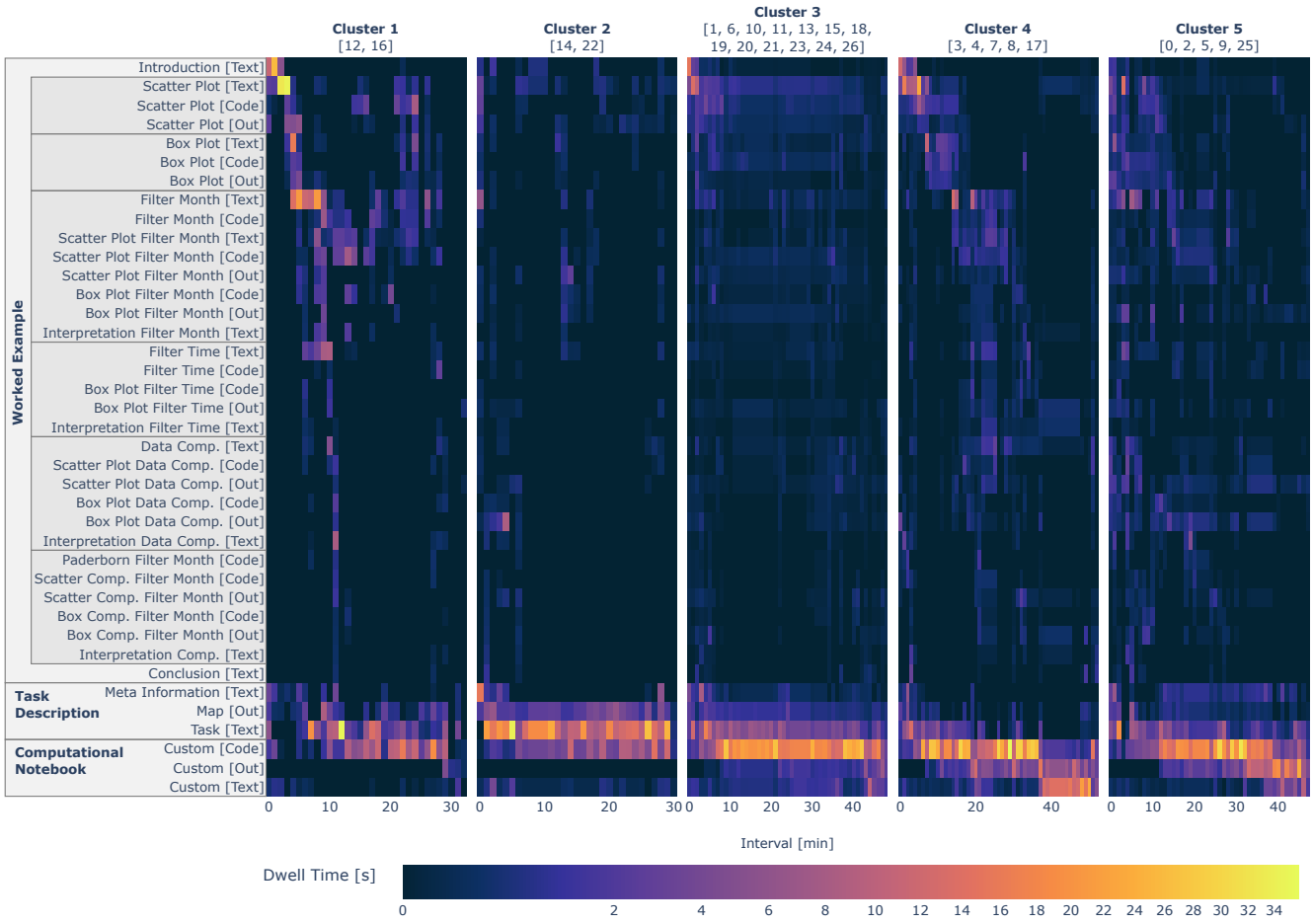


Figure 3: Aggregated clusters using DTW Barycenter Averaging (DBA). Values were plotted on a non-linear colour scale using the square root of the actual value. This was done to increase contrast also for cells that were only briefly fixated.

Table 1: Performance metrics for each cluster, including the number of participants (n), the mean trial duration and the number of participants from each cluster who produced at least one basic visualisation (scatter plot, bar plot), one combined visualisation (multiple traces), applied a data filter, and based their reasoning on their results.

Cluster	n	Duration [min] M	Duration [min] SD	Basic visualisation	Combined visualisation	Filter	Reasoning based on results
1	2	31.95	0.43	2 (100%)	1 (50%)	1 (50%)	1 (50%)
2	2	24.66	8.89	1 (50%)	0 (0%)	0 (0%)	2 (100%)
3	13	26.26	9.09	13 (100%)	9 (69%)	2 (15%)	10 (77%)
4	5	44.83	9.64	5 (100%)	5 (100%)	5 (100%)	5 (100%)
5	5	42.59	7.89	5 (100%)	4 (80%)	3 (60%)	4 (80%)

For cluster 3 ($n = 13$), two linear reading phases can be identified in the heat-map - one directly at the beginning, which appears to be merely a scanning phase, and the other one after approx. 30 minutes. There were longer dwell times in the sections for the scatter plot as well as less intense, but also longer phases in the sections for the box plot, filter month, filter time and data comparison. The participants in this cluster seemed to have spent more time on these individual aspects. Regarding dwell times on custom code, there were additionally longer intense phases beginning after approx. 8 minutes. Hence, the participants appear to have spend a lot of time working on and reviewing their code. Additionally, after approx. 40 minutes there were parallel or integrated phases with higher

dwell times in the custom output and text cells. Regarding their products, most participants arrived at a decision that was justified on the basis of the previously obtained results. Additionally, most of them were able to create a combined visualisation for comparing the data from the different cities.

Participants in cluster 4 ($n = 5$) read through the worked example in a linear order, while simultaneously looking at their own code from the 5th minute onwards. However, there seemed to be a harsh cut after approx. 35 minutes, when they shifted their attention towards the custom output and the custom markdown text. Additionally, it is noticeable that the participants from this cluster did not or only shortly revisit the cells in the worked example that they had visited before. In their Jupyter notebook, all participants from cluster 4 created at least three visualisations including a combined visualisation, applied at least one filter and were able to arrive at a decision, connected with insights from their programming results.

The participants from cluster 5 ($n = 5$) showed a similar behaviour to the participants from cluster 4. While the heat-map implies that they also had one rather short linear run-through of the worked example and afterwards looked into the worked example and their own code cells simultaneously (especially approx. between minutes 12 and 38), they apparently looked into their custom code more intensively after approx. 25 minutes. Compared with cluster 4, they intensively looked into their own output and text, while they were still looking into their custom code. However, they did not look into their own custom text as intensively. Nevertheless, each participant from this cluster created at least two visualisations, while three of them also applied at least one filter. All but one participant created a combined visualisation and connected the final decision with insights gained from the programming process.

Apparently, the participants exhibited diverse strategies during their programming processes. Across all clusters, participants engaged in at least one "scanning" phase of the worked example in the beginning of their process.

While clusters 1 and 4 overall exhibited one rather linear navigation through the individual sections, the participants from the other clusters apparently had multiple walk-troughs (cluster 2) or focused on specific aspects, not necessarily in the sequence, shown in the worked example (cluster 3 and 5). Additionally, while the participants from clusters 1, 2, 3 and 5 apparently revisited certain cells, participants from cluster 4 seem to have only rarely returned to cells they had already visited, so they probably completed their programming process regarding the respective aspect first, before moving on.

Furthermore, variations in code-related behaviours were observed. For instance, participants in clusters 1, 2, and 5 predominantly began coding activities after a linear walk-through, whereas those in clusters 3 and 4 demonstrated earlier and more intertwined coding, while simultaneously looking into the worked example.

Also regarding participants' consideration of their own output and text cells, diverse behaviours could be identified. In particular, it is noticeable that in cluster 4, the phases of writing code and writing text were apparently separated, which does not seem to be the case in the other clusters. Regarding the review of the students' Jupyter notebooks (as it can be seen from Table 1), cluster 4 emerged as the most successful. Looking into the heat-map in Figure 3, participants from this cluster seemingly demonstrated a focused examination of specific sections within the provided worked example, which they might deemed helpful for their epistemic programming endeavours. This targeted approach mirrors the described expert-strategies described by Schoenfeld [Schoenfeld 2016], which Reiss & Renkl [Reiss and Renkl 2002] took up with the idea of heuristic worked examples. This, in turn suggests that participants in cluster 4 effectively leveraged heuristic worked examples to navigate the problem space and inform their decision-making processes, which might be characteristic for epistemic programming.

Based on the results, our method of clustering and aggregating scan paths using DTW and DBA appears promising, especially for smaller cluster sizes, as it provides a visualisation of the aggregated clusters that is both meaningful and easy to interpret. We believe that it is particularly useful for long-running experiments to be able to distinguish between different process stages and to determine the order in which they occur. However, as information about transitions is partially lost, it is not useful for detecting local scan patterns. In such a case, it is recommended to examine the AOI sequences and use techniques such as multiple sequence alignment to compare the flow of attention.

6 LIMITATIONS

As stated in section 3.3, some of the participants' fixations on the output were incorrectly mapped to the corresponding code cells due to a bug in the version of JuGaze that was used. However, we assume that this did not affect the results to a significant extent. Additionally, the data of some participants apart from the data, we used here, were recorded incorrectly. However, we were still able to retrieve complete data sets for 26 participants. The resulting clusters vary significantly in size, spanning from 2 to 13 participants. As more participants are added to the clusters, they tend to become increasingly blurred and the sequence length increases, potentially posing challenges for compatibility. Consequently, the generalisability of the results is limited. Finally, it should be noted that while DTW takes into account time shifts and different speeds between participants, it is not suitable for detecting recurring gaze patterns (e.g. loops).

7 CONCLUSION & FUTURE WORK

The aim of this study was to identify different approaches of novices in using worked examples in the context of epistemic programming endeavours. For this purpose, we used Dynamic Time Warping to cluster and aggregate individual scan paths, which allowed us to identify common gaze patterns between participants.

We found that there were different approaches to the use of the worked examples in terms of the order in which the cells were fixated and the way in which the programming was interwoven with the utilisation of the worked example. Drawing from our analysis, we hypothesise that students might use worked examples in order to gain an initial idea of possible steps and actions as well as regarding a more detailed examination of these later on. Concurrent reference to the worked example throughout the programming process seems especially effective (cf. cluster 4), where students can obtain information about the current aspect they are working on. Additionally, we argue that certain approaches show characteristics of epistemic

programming practices, involving a selective adaptation of the provided code snippets based on the individual interest, rather than a mere reproduction of the worked example. This also includes an intertwined programming and reading process. However, we do not know whether this is because these participants' approach was better suited to the given setting, or because it was different due to differences in their programming skills. For a future study to address this question, it might be useful to develop an instrument to determine the successful adoption of epistemic programming regarding the exploration of individual areas of interest.

Generally, future findings could be relevant for designing worked examples, as they indicate different types of learners that need to be considered. In addition, students could also be given guidance on utilising worked examples by explicitly teaching "successful" approaches. Overall, this study has already been able to identify different approaches regarding the utilisation of worked examples in epistemic programming endeavours. The methodology used additionally provides a basis for concretising and identifying further approaches in the future.

ACKNOWLEDGMENTS

This work was created as part of the ProDaBi project (https://www.prodabi.de), funded by the Deutsche Telekom Stiftung and as part of the CDEC project (https://cdec.io), funded by the Bundesministerium für Wirtschaft und Klimaschutz.

REFERENCES

Robert K. Atkinson, Sharon J. Derry, Alexander Renkl, and Donald Wortham. 2000. Learning from Examples: Instructional Principles from the Worked Examples Research. *Review of Educational Research* 70, 2 (June 2000), 181–214. https://doi.org/10/csm67w

Roman Bednarik and Markku Tukiainen. 2006. An Eye-Tracking Methodology for Characterizing Program Comprehension Processes. In *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications - ETRA '06*. ACM Press, San Diego, California, 125. https://doi.org/10.1145/1117309.1117356

Stephan A. Brandt and Lawrence W. Stark. 1997. Spontaneous Eye Movements during Visual Imagery Reflect the Content of the Visual Scene. *Journal of Cognitive Neuroscience* 9, 1 (1997), 27–38. https://doi.org/10.1162/jocn.1997.9.1.27

Michael Burch, Kuno Kurzhals, Niklas Kleinhans, and Daniel Weiskopf. 2018. EyeMSA: Exploring Eye Movement Data with Pairwise and Multiple Sequence Alignment. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*. ACM, Warsaw Poland, 1–5. https://doi.org/10.1145/3204493.3204565

Teresa Busjahn, Carsten Schulte, Bonita Sharif, Simon, Andrew Begel, Michael Hansen, Roman Bednarik, Paul Orlov, Petri Ihantola, Galina Shchekotova, and Maria Antropova. 2014. Eye tracking in computing education. In *Proceedings of the tenth annual conference on International computing education research*. ACM, Glasgow Scotland United Kingdom, 3–10. https://doi.org/10.1145/2632320.2632344

Andrea A. DiSessa. 2000. *Changing minds: computers, learning, and literacy*. MIT Press, Cambridge, Mass.

Sukru Eraslan, Yeliz Yesilada, and Simon Harper. 2016. Scanpath Trend Analysis on Web Pages: Clustering Eye Tracking Scanpaths. *ACM Transactions on the Web* 10, 4 (Dec. 2016), 1–35. https://doi.org/10.1145/2970818

Joseph H. Goldberg and Jonathan I. Helfman. 2010. Scanpath Clustering and Aggregation. In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications - ETRA '10*. ACM Press, Austin, Texas, 227. https://doi.org/10.1145/1743666.1743721

Sven Hüsing, Carsten Schulte, Sören Sparmann, and Mario Bolte. 2024. Using Worked Examples for Engaging in Epistemic Programming Projects. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. ACM, New York, NY, USA. https://doi.org/10.1145/3626252.3630961

Sven Hüsing, Carsten Schulte, and Felix Winkelnkemper. 2023. Epistemic Programming. In *Computer Science Education: Perspectives on Teaching and Learning in School* (2 ed.), Sue Sentance, Erik Barendsen, Nicol R. Howard, and Carsten Schulte (Eds.). Bloomsbury Academic, London, 291–304. http://dx.doi.org/10.5040/9781350296947.ch-022

Marcel Adam Just and Patricia A Carpenter. 1976. Eye fixations and cognitive processes. *Cognitive Psychology* 8, 4 (Oct. 1976), 441–480. https://doi.org/10.1016/0010-0285(76)90015-3

Pawel Kasprowski and Katarzyna Harezlak. 2019. Using Mutual Distance Plot and Warped Time Distance Chart to Compare Scan-Paths of Multiple Observers. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*. ACM, Denver Colorado, 1–5. https://doi.org/10.1145/3317958.3318226

A Ben Khedher, Imène Jraidi, and Claude Frasson. 2018. Local sequence alignment for scan path similarity assessment. *International Journal of Information and Education Technology* 8, 7 (2018), 482–490.

Ayush Kumar, Rudolf Netzel, Michael Burch, Daniel Weiskopf, and Klaus Mueller. 2018. Visual Multi-Metric Grouping of Eye-Tracking Data. *Journal of Eye Movement Research* 10, 5 (Feb. 2018). https://doi.org/10.16910/jemr.10.5.10

Ayush Kumar, Neil Timmermans, Michael Burch, and Klaus Mueller. 2019. Clustered Eye Movement Similarity Matrices. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*. ACM, Denver Colorado, 1–9. https://doi.org/10.1145/3317958.3319811

Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *ACM Inroads* 2, 1 (Feb. 2011), 32–37. https://doi.org/10/ggdnzr

Vladimir I Levenshtein et al. 1966. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. In *Soviet Physics Doklady*, Vol. 10. Soviet Union, 707–710.

Yvonne G. Mulder, Ard W. Lazonder, and Ton De Jong. 2014. Using heuristic worked examples to promote inquiry-based learning. *Learning and Instruction* 29 (Feb. 2014), 56–64. https://doi.org/10.1016/j.learninstruc.2013.08.001

Kasia Muldner, Jay Jennings, and Veronica Chiarelli. 2023. A Review of Worked Examples in Programming Activities. *ACM Transactions on Computing Education* 23, 1 (March 2023), 1–35. https://doi.org/10.1145/3560266

Unaizah Obaidellah, Mohammed Al Haek, and Peter C.-H. Cheng. 2019. A Survey on the Usage of Eye-Tracking in Computer Programming. *Comput. Surveys* 51, 1 (Jan. 2019), 1–58. https://doi.org/10.1145/3145904

Tor Ole B. Odden, Devin W. Silvia, and Anders Malthe-Sørenssen. 2022. Using computational essays to foster disciplinary epistemic agency in undergraduate science. *Journal of Research in Science Teaching* 60, 5 (2022), 937–977. https://doi.org/10.1002/tea.21821

Fernando Perez and Brian E. Granger. 2015. Project Jupyter: Computational Narratives as the Engine of Collaborative Data Science. https://blog.jupyter.org/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science-2b5fb94c3c58

François Petitjean, Alain Ketterlin, and Pierre Gançarski. 2011. A Global Averaging Method for Dynamic Time Warping, with Applications to Clustering. *Pattern recognition* 44, 3 (2011), 678–693. https://doi.org/10.1016/j.patcog.2010.09.013

Siti Soraya Abdul Rahman and Benedict du Boulay. 2010. Learning programming via worked-examples. *PPIG-WIP, Dundee 2010* (2010), 1–6.

Kristina Reiss and Alexander Renkl. 2002. Learning to prove: The idea of heuristic examples. *Zentralblatt für Didaktik der Mathematik* 34, 1 (Feb. 2002), 29–35. https://doi.org/10.1007/BF02655690

Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education* 13, 2 (June 2003), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

H. Sakoe and S. Chiba. 1978. Dynamic Programming Algorithm Optimization for Spoken Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26, 1 (1978), 43–49. https://doi.org/10.1109/TASSP.1978.1163055

Dario D. Salvucci and Joseph H. Goldberg. 2000. Identifying Fixations and Saccades in Eye-Tracking Protocols. In *Proceedings of the Symposium on Eye Tracking Research & Applications - ETRA '00*. ACM Press, Palm Beach Gardens, Florida, United States, 71–78. https://doi.org/10.1145/355017.355028

Alan H. Schoenfeld. 2016. Learning to Think Mathematically: Problem Solving, Metacognition, and Sense Making in Mathematics (Reprint). *Journal of Education* 196, 2 (April 2016), 1–38. https://doi.org/10.1177/002205741619600202

Sören Sparmann, Sven Hüsing, and Carsten Schulte. 2023. JuGaze: A Cell-based Eye Tracking and Logging Tool for Jupyter Notebooks. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. ACM, Koli Finland, 1–11. https://doi.org/10.1145/3631802.3631824

John Sweller, Paul Ayres, and Slava Kalyuga. 2011. *Cognitive Load Theory*. Springer New York, New York, NY. https://doi.org/10.1007/978-1-4419-8126-4

John Sweller and Graham A. Cooper. 1985. The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra. *Cognition and Instruction* 2, 1 (March 1985), 59–89. https://doi.org/10.1207/s1532690xci0201_3

John Sweller, Jeroen J. G. van Merrienboer, and Fred G. W. C. Paas. 1998. Cognitive architecture and instructional design. *Educational Psychology Review* 10, 3 (1998), 251–296. https://doi.org/10.1023/A:1022193728205

Uri Wilensky, Corey E. Brady, and Michael S. Horn. 2014. Fostering computational literacy in science classrooms. *Commun. ACM* 57, 8 (Aug. 2014), 24–28. https://doi.org/10/gqv6bn

Stephan Wolfram. 2017. What Is a Computational Essay? https://writings.stephenwolfram.com/2017/11/what-is-a-computational-essay/