Chapter # - will be assigned by editors

VISUALIZATION OF STUDENT-ITEM INTERACTION MATRIX

Tomáš Effenberger, Masaryk University, tomas.effenberger@mail.muni.cz

Radek Pelánek, Masaryk University, pelanek@fi.muni.cz

Abstract: One type of visualization of data from digital learning environments focuses on students' interaction with the educational content. Students may, for example, answer questions, read texts, or solve problems. We can represent these interactions as a matrix, where rows correspond to students, columns to educational items, and values to some aspect of student activity (e.g., the correctness of answers, response times, the order of actions). Visualizing this matrix is useful for several purposes. For teachers, it can provide an understanding of the skill and behavior of their students. For system developers, it can provide insight into the behavior of both students and adaptive algorithms, and it can also help detect suspicious activity. For researchers, it can provide an understanding of the properties of datasets used in experiments and valuable warnings about biases that are present in data. However, suitable visualization of the student-item interactions is nontrivial. To facilitate the design of the visualization, we provide a systematic discussion of approaches to student-item matrix visualization. Using data from an introductory programming exercise, we also provide specific illustrations of different visualization designs.

Key words: exploratory data analysis, learning environment, heatmap, dotted chart

1. INTRODUCTION

In digital learning environments, students do not just passively consume learning content but also actively interact with various educational items. In this work, we focus on visualizing these interactions. We consider *item* as a general term encapsulating, among others, multiple-choice questions, fill-in-the-blank and drag-and-drop exercises, interactive simulations, or programming assignments. We consider particularly items for which the student interaction can be automatically evaluated as correct or incorrect.

Data on student-item interaction are valuable for many stakeholders. For students and teachers, the data can provide insight into the state of the learning process. For developers of learning environments, the data can give impulses for the improvement of their environments. For researchers, the data can provide inspiration for the design and evaluation of personalization algorithms.

Student-item interaction can be analyzed and visualized in many ways. To put the techniques discussed in this work into a context, it is useful to consider whether the visualization is concerned with single or multiple students and items. Figure 1 provides illustrations for different combinations. The figure uses simplified versions of visualizations and hypothetical data about student interaction with programming items.

- *Single student, single item.* The most detailed visualization, focusing on individual actions of a single student while solving an item. The provided example visualizes student's edits and submits while solving a programming assignment.
- *Single student, multiple items.* This visualization shows the behavior of a single student across multiple items, e.g., as a bar chart showing the activity of a given student through time. Other examples of this type of visualization are MasteryGrid without social features by Brusilovsky et al. (2016), the summative visualization of student activity using Chernoff faces (France, Heraud, Marty, Carron, &

Heili, 2006), and line chart displaying progress and responses to test items in chronological order (Costagliola, Fuccella, Giordano, & Polese, 2008).

- *Multiple students, single item.* For a single item, we can visualize the activity of multiple students to show different approaches to solving the item. This can be done, for example, using an interaction network with nodes representing possible partial solutions and edges representing frequent transitions between them (Johnson, Eagle, & Barnes, 2013).
- *Multiple students, multiple items.* Finally, we can consider both multiple students and multiple items, which naturally leads to a matrix-based visualization. This approach is the focus of this chapter.

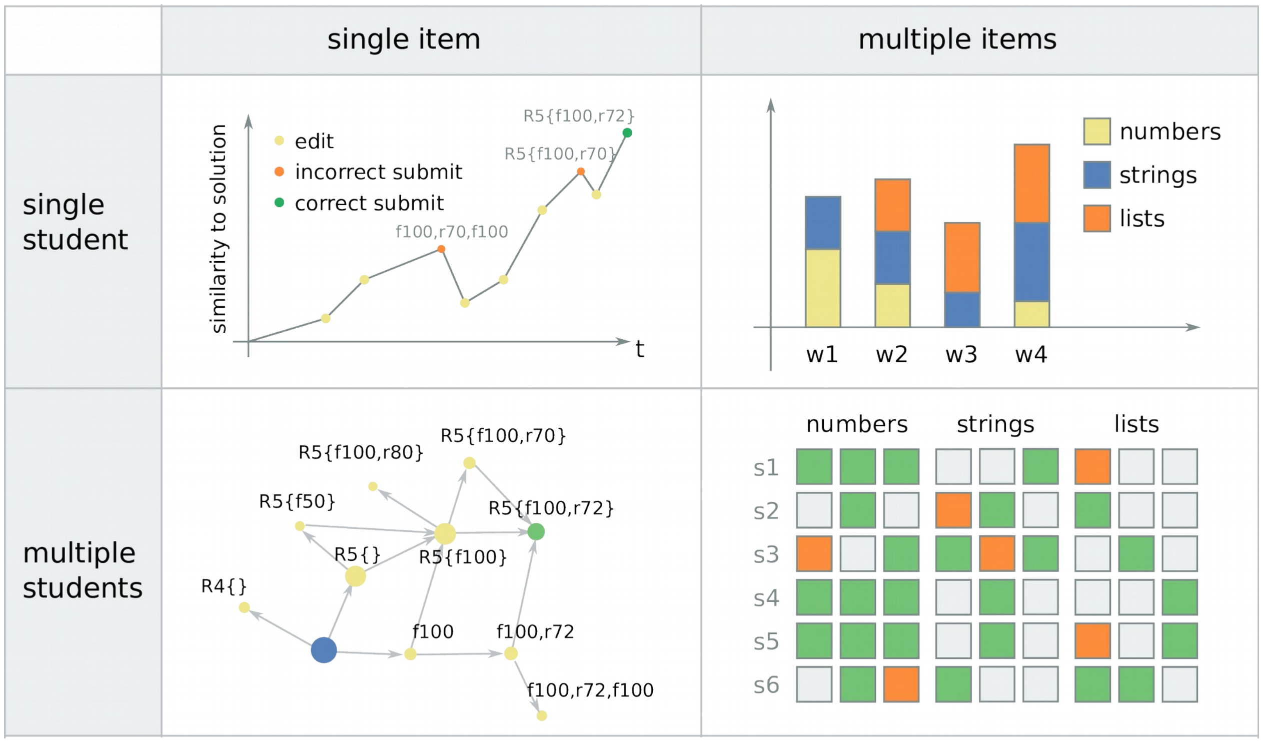

Figure 1. Examples of visualizations of student-item interactions depending on the number of students and items.

We focus on student-item interaction visualization taking into account multiple students and multiple items, particularly in the form of a *student-item matrix* with rows and columns corresponding to students and items. This type of visualization can be found in the literature under various other names: student-problem matrix (Khajah, Wing, Lindsey, & Mozer, 2014), student-problem chart (Wang & Chen, 2013), lesson overview (Molenaar & Knoop-van Campen, 2017), and even just heat map (Confrey, Gianopulos, McGowan, Shah, & Belcher, 2017). None of these publications provides a systematic discussion of the student-item matrix. Each of them uses one particular variant of the student-item matrix (e.g., students and items ordered by their skill and difficulty, cells displaying binary correctness) for a specific purpose (e.g., providing feedback to teachers about struggling students).

Analogous matrix visualization has been used to display the interaction of students with other entities as well. For instance, the columns can correspond to knowledge components (Brusilovsky et al., 2016; Mazza & Dimitrova, 2007), courses (A. J. Bowers, 2010), errors (Fu, Shimada, Ogata, Taniguchi, & Suehiro, 2017), or types of learning activities (Lee, Recker, Bowers, & Yuan, 2016).

Another variation on the standard student-item matrix is to use time as the horizontal axis, resulting in a sparse *dotted chart* instead of the dense heat map (Aalst, Guo, & Gorissen, 2013; Sedrakyan, Snoeck, &

De Weerdt, 2014; Trcka, Pechenizkiy, & Aalst, 2010). Such visualization can be considered as a (non-standard) student-item matrix only if the cells still correspond to individual student-item interactions. Aggregating the interactions, e.g., as the number of interactions per day, can be useful, but we do not call such visualization a student-item matrix since we cannot make inferences about the individual items.

Other research areas use closely related techniques. In recommender systems, a key data structure is a *user-item matrix* with ratings (e.g., movie ratings); Monti, Rizzo, & Morisio (2019) uses 3D visualization of this matrix. In process mining, similar methods are used to visualize a *resource-activity matrix* (Janssenswillen, Depaire, Swennen, Jans, & Vanhoof, 2019).

This chapter presents a systematic treatment of the visualization of student-item matrices. The student-item matrix has many applications (Section 2) and each application leads to specific requirements that should be taken into account when designing the visualization. Although several studies already used this visualization, they do not focus on the student-item matrix per se and do not provide any guidance to its design. We provide a systematic discussion of different aspects of the visualization and also describe variations and extensions (Section 3). The chapter includes a case study with data from introductory programming, which illustrates different variants of the student-item matrix and discusses the insights they provide (Section 4).

2. APPLICATIONS

The general aim of visualizing the student-item matrix is to get an understanding of data and insights into underlying behaviors and consequently to make informed decisions that will lead to the improvement in student learning. More specifically, the goal is to understand the behavior of students, algorithms, and their interactions. This is nontrivial since student's behavior is complex and noisy, personalization algorithms are adaptive, and interactions can have surprising effects.

The visualization can serve several different specific needs, depending on the target audience (students, teachers, developers, researchers). We outline several of these applications. To illustrate them, we use the matrix shown in Figure 2. This example visualizes student answers to a reading comprehension exercise, where a student reads a text and then answers several multiple-choice questions. The matrix is hypothetical, i.e., it is not based on real student data but artificially constructed in order to show in a compact space many potential applications. Nevertheless, all the discussed patterns are based on our experiences with real data. Real data are, of course, noisier and we do not see so many aspects of student behavior in such a small sample.

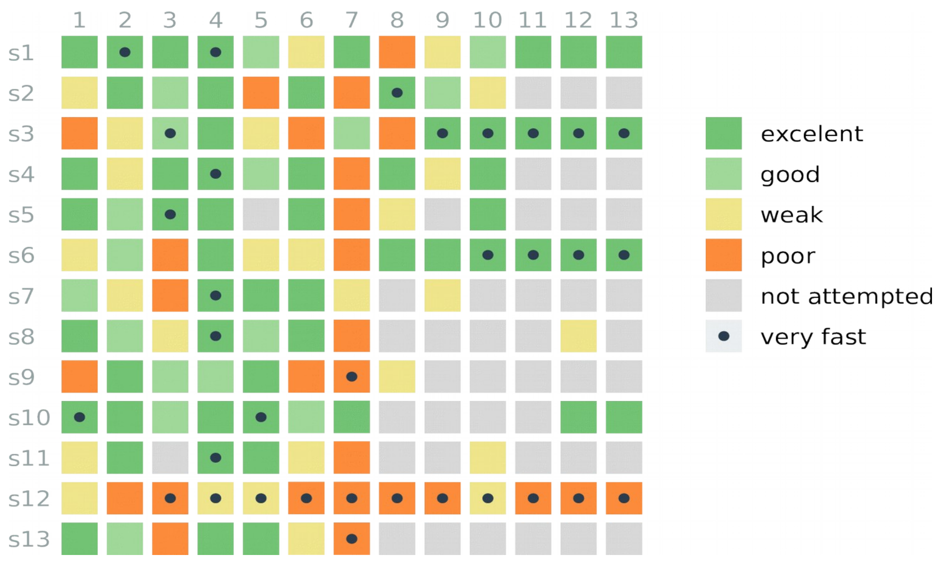

Figure 2. Hypothetical student-item matrix for reading comprehension exercise. The color is based on the correctness of student answers.

2.1 Feedback to students and teachers

Teachers can use the visualization during a class to decide what to do, e.g., to give feedback to a particular student or to discuss something with the whole class (Confrey et al., 2017; Molenaar & Knoop-van Campen, 2017). In our illustrative example in Figure 2, a teacher may quickly conclude that student s1 works mostly well, student s11 needs help, and student s12 is completely disengaged. Molenaar et al. (2017) describe how teachers use the student-item matrix and other visualizations during class; they confirm that these visualizations influence their actions.

The visualization can be incorporated into an open learner model in order to help students to develop metacognitive skills while simultaneously serving as navigation through the system (Brusilovsky et al., 2016).

2.2 Understanding behavior, decision support

Student-item matrix visualization can provide system developers and content authors with an understanding of student behavior. How are students interacting with the content? Is the interaction as expected? Can we detect different types of students? Do we need to modify or extend the available content (e.g., add more items or add easier items)? Consider the illustration in Figure 2. Here we can see that item 4 is probably too easy, whereas item 7 is very difficult and many students stop the practice at this item. This is clearly a critical point that requires attention and suitable modification.

Previous work used related techniques and visualizations with similar aims, e.g., detection of student clusters based on sequences of their actions (Desmarais & Lemieux, 2013), visualization of data about student behavior in MOOC courses (Coffrin, Corrin, Barba, & Kennedy, 2014), or visualization of data about player behavior in games (Wallner & Kriglstein, 2013).

2.3 Detecting counterproductive behavior

Students do not always use learning environments in the productive fashion intended by designers. There are many types of counterproductive behavior, e.g., cheating, systematic guessing, or gaming-the-system (hint abuse) (Baker et al., 2008; Northcutt, Ho, & Chuang, 2016). There are many types of counterproductive behavior and students are often surprisingly creative—we have, for example, encountered cases of intensive exploration of HTML source code or JavaScript console outputs. Visualizations can often provide indications of suspicious activity (Costagliola et al., 2008). Once we spot unexpected patterns in the visualization, we can build detectors to quantify them and find them systematically.

In the illustration in Figure 2, students s3 and s6 are probably cheating. At the beginning of the sequence, they struggle to answer items correctly. At the end of the sequence, they have a long sequence of excellent and very fast answers. We can also see that student s12 is just guessing, which is another form of counterproductive behavior.

2.4 Understanding biases in data

The data from learning environments are typically skewed and may contain various biases (Nixon, Fancsali, & Ritter, 2013; Čechák & Pelánek, 2019), e.g., mastery attrition bias (students who know a topic are leaving earlier than weak students) or ordering bias (items presented at the beginning of a sequence are solved by many more students and under different circumstances than items presented later). These biases can significantly influence the evaluation of student models and learning environments (Pelánek, 2018). Visualizations can help us understand the biases and skews present in a particular dataset and to make informed decisions concerning the proper evaluation methodology, e.g., splitting the dataset into a training and testing set or the approach to the computation of metrics.

The illustration in Figure 2 shows a typical skew in the distribution of answers due to item order. Items 9–13 are solved by only a small subset of students and these students are not a representative sample (only good or cheating students).

2.5 Inspiration and intuition for student models

Learning environments often provide adaptive behavior that is guided by student modeling (Pelánek, 2017). Based on student activity, a student model provides an estimate of a student state. Student models can take many forms and use many types of input data; the choice of a suitable model depends on a specific situation. Visualizations of interaction can often provide guidance and inspiration for the design of student models. For example, in data from one system, we noticed quite frequent consecutive sequences of incorrect answers. Based on this observation, we built a simple predictor of next answer correctness, which was competitive with more sophisticated student models (Řihák & Pelánek, 2016). A specific application is the choice of performance data to use. There are many aspects of student performance that can be used in student modeling (e.g., the correctness of answers, response times, the quality of solutions, absolute timestamp, class membership). Student-item matrix can capture multiple aspects of performance, providing insight into how these performance aspects are related, which of them are noisy and which carry a consistent signal about the student. It is useful to have intuition before one plunges into modeling. For example, for the data depicted in Figure 2, it seems that response times may be indicative of affective state (disengagement) and cheating but probably would not be very useful for modeling cognitive state (at least without nontrivial filtering).

The student-item matrix visualization is also used for the illustration of methodological issues in student model evaluation (train-test data split) (Khajah et al., 2014; Pelánek, 2018; Reddy, Labutov, Banerjee, & Joachims, 2016). After performing the modeling and evaluation, visualizations can be useful for checking the validity of results and providing interpretation.

3. DESIGN OF THE MATRIX VISUALIZATION

The student-item matrix is similar to heatmaps and scatterplots but requires additional decisions concerning filtering, grouping, and ordering of the students and items. A large number of parameters makes the student-item matrix a rich and versatile visualization but can be intimidating the first time you use it. To help design a suitable student-item matrix, this section provides a systematic overview of the parameters and available options.

3.1 Standard student-item matrix

In the standard student-item interaction matrix, rows correspond to individual students, columns to individual items, and cells to interactions between them. Each of these three graphical components — rows, columns, and cells — has a number of parameters, which are shown in Figure 3. In the following, we discuss typical options for these parameters.

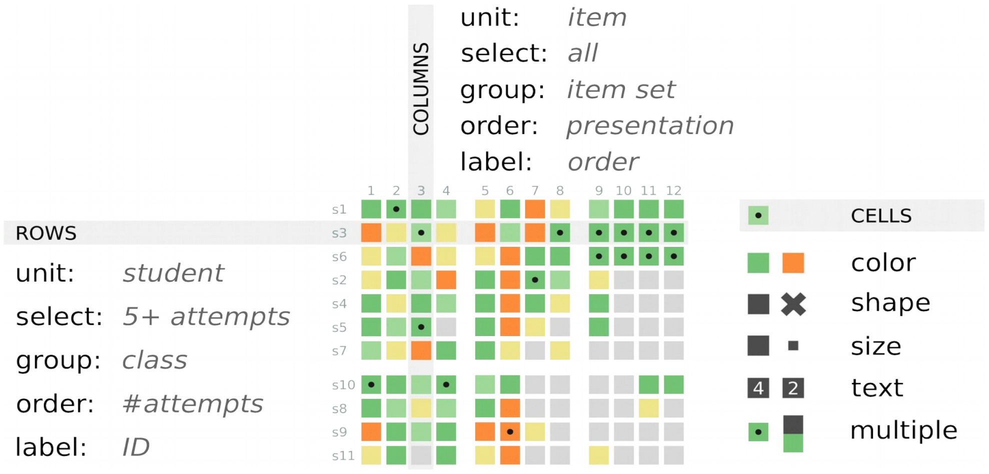

Figure 3. Parameters of rows, columns, and cells in student-item interaction matrix, together with an example set of options (printed in italics).

3.1.1 Rows — students

There might be several orders of magnitude more students than we can fit into the visualization, so we must choose just a subset of them. In addition to the selection of students, the second key parameter is the ordering, which can often greatly enhance the intelligibility of the visualization.

- **unit**: *individual student, group of students (class, cluster)*
  Typically, we want to see individual students, but lower granularity is certainly possible. Each row would then represent a set of students, e.g., a cluster of students with similar behavior.
- **select**: *filter by condition, random sample, top N*
  First, we can filter students satisfying a specific condition, e.g., students from a specific class or students who attempted at least 10 items. Then — if there are still too many students — we take a random sample, possibly stratified (e.g., an equal number of male and female students) or blocked (e.g., students with activity within one randomly chosen week). Alternatively, we can select top *N* students with respect to a criterion such as the number of answers.
- **group**: *grade, class, school, or another categorical attribute*

If we want to compare multiple groups of students, we can put the rows of the students from the same group together and insert a small gap (or simply an empty column) between the groups.

- **order**: *activity, skill, or another numerical attribute*
  A reasonable default choice is a summary of a student's activity, such as the number of interactions, success rate, or a skill estimated by a student model. Sometimes, other orderings are more appropriate. For instance, ordering by the time of students' first (or last) activity may reveal group cheating or the impact of new items. A more sophisticated way to put similar students close to each other is to define the similarity between two students and use 1D dimensionality reduction or dendrogram resulting from hierarchical clustering (Lee et al., 2016).
- **label**: *ID, name, or another categorical attribute*
  Identifying individual students is important when the matrix is used as an overview for a teacher, but for most of the other applications, labels are not needed.

3.1.2 Columns — items

There are typically much fewer items than students, so it might be feasible to show all of them. If not, we can either aggregate them to larger units or select just a subset of items. As for the students, the ordering of items is an important decision.

- **unit**: *individual items, steps, item sets, knowledge components, courses*
  The default choice are individual items, but both lower and higher granularity is possible; e.g., units of higher granularity are steps within an item, units of lower granularity are item sets.
- **select**: *all, filter by condition, random sample*
  If there are too many items, we can select a group of closely related items such as an item set or a knowledge component. Alternatively, we can use a random sampling strategy, analogically as for students.
- **group**: *item set, type of item, or another categorical attribute*
  There are often natural groups of items, such as item sets or item types (e.g., multiple-choice vs. free-response questions). It might be helpful to visually separate these groups.
- **order**: *presentation order, difficulty, or another numerical attribute*
  If the items have some predetermined ordering within the system, it is natural to use the same ordering also in the student-item matrix. For some use cases, alternative orderings might make sense, e.g., by difficulty (e.g., success rate) or by the time when the item was created. Using per-student ordering of items is possible (e.g., in the order they solved the items), but then columns do not correspond to unique items; this is discussed separately in section 3.2.
- **label**: *ID, or another categorical attribute*
  There is not much space in the header for each item. If we want to show the names (or even complete item statement), we need to either rotate the labels, transpose the matrix (i.e., dedicate rows to items), or use interactive features such as mouse hover.

3.1.3 Cells — interactions

Given a student-item pair, we can display various data about their interaction:

- student's performance (e.g., correctness, solution quality, response time, the number of attempts or requested hints),

- time of the interaction (e.g., date, the time within a day, the time from the first student's interaction, the order of the student's interaction),
- prediction of a student model (e.g., the predicted probability that the student solves the item, that he is frustrated, or that he is cheating).

These *data attributes* can be mapped to any subset of the cells' *graphical attributes* shown in Figure 3: color, shape, size, and text. The graphical and data attributes are nearly orthogonal and can be combined in many ways. One constraint is that some graphical attributes are only suitable for categorical data attributes. It is, however, possible to discretize a numerical attribute into a few categories, e.g., using "short/long solution" instead of the continuous length of the solution.

- **color**: *performance, time, any categorical or numerical attribute*
  Changing the color is the least disruptive way to vary the appearance of the cells without making the matrix more difficult to navigate. Color can represent both categorical data (using a qualitative colormap) and numerical data (using a sequential colormap). It is even possible to show multiple performance aspects, either by mapping separately hue and lightness to two different aspects (e.g., correctness and response time), or by combining multiple aspects into a single category, e.g., "weak performance" when either the response time is high or the quality of the solution is low. Recommendations on how to perform such answer classification in various domains exist (Pelánek & Effenberger, 2020).
- **shape**: *suspicious behavior, correctness, or another categorical attribute*
  If the matrix is dense, using multiple shapes would produce unintelligible visualization. This graphical attribute can be useful if there are a few interactions that we want to highlight, e.g., detected cheating. Another example might be using crosses for serious unfinished attempts in a problem-solving exercise where nearly all of the items are eventually solved.
- **size**: *response time, or another numerical attribute*
  We can either change just the width or height of the shape or both dimensions simultaneously. For example, we can scale crosses representing unfinished attempts proportionally to the response time in order to make the non-serious attempts less prominent.
- **text**: *item ID, or another categorical or numerical attribute*
  If there are not many interactions and the cells are large enough, we might be able to fit a short text (letter, 2-digit number) in each cell. However, in a more typical scenario, there are many interactions and the cells are thus too small. A possible remedy is to use interactive features — showing the text on mouse hover, click, or after sufficient zoom-in.
- **multiple**: *composing graphical attributes, nesting shapes, stacking cells*
  Often, we want to directly compare two data attributes, e.g., two aspects of performance, predicted vs. actual performance, or the performance vs. the difficulty of the item. The most obvious way to show multiple data attributes for each interaction is to vary multiple graphical attributes of the cells. For instance, a shape can denote correctness and color response time. There are two other approaches, which might better preserve the grid regularity: nesting and stacking. Figure 2 illustrates nesting multiple shapes in a single cell. The nested shape might not be just a binary indicator; it can possess any of the discussed graphical attributes.

3.2 Non-standard student-item matrix

In the standard version of the student-item interaction matrix, each row corresponds to a unique student, each column to a unique item, and each cell to the interaction between them. If we drop the requirement on the rows and columns but insist that each cell still corresponds to a student-item interaction, we obtain a much broader set of visualizations, which we call *non-standard* student-item interaction matrices.

A prominent class of non-standard student-item interaction matrices uses the x-axis to display time instead of to identify the items. Such visualization is called *dotted chart* in the process mining community (Janssenswillen et al., 2019; Song & Aalst, 2007). It is useful when the temporal aspect is important, e.g., for debugging a student model or investigating possible cheating before the homework deadline.

There are many notions of time to consider, and the appropriate choice depends on the specific application. There is a fundamental trade-off between the fidelity of the time-axis and the compactness of the visualization, which is illustrated in Figure 4. Three basic choices are absolute, relative, and logical time.

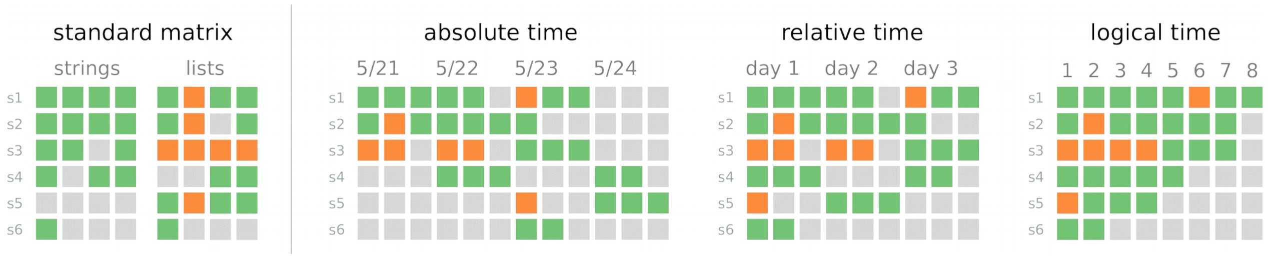

Figure 4. Comparison of standard and non-standard student-item interaction matrices. The non-standard versions use three different notions of time for the x-axis.

- **absolute time**: The columns represent discretized absolute time. To avoid overlapping interactions, we may use just absolute dates, using the specific time only to order the interactions within the day. To avoid too wide visualization, we can decrease the width of each cell; this strategy is used in Figure 9.
- **relative time**: Absolute time becomes impractical if the times for the set of selected students differ widely. In such a case, we might use time relative to a given student, e.g., *n*th day since the student's first interaction.
- **logical time**: The most compact — and least faithful — visualization is obtained by keeping only the information about the ordering of the interactions, i.e., *n*th column corresponds to *n*th interaction for a given student. The resulting time-ordered student-item interaction matrix is dense, with no empty cells between interactions; see Figures 7 and 8.

Other variants of non-standard student-item interaction matrices are possible but much less frequently useful. For instance, if new content was added to our learning system and we want to explore how this change impacted performance on the existing items, we could use rows to identify items (instead of students) and the x-axis to display time before and after the content update.

3.3 Extensions

Figure 5 shows examples of additional graphical elements that can be added to the student-item interaction matrix.

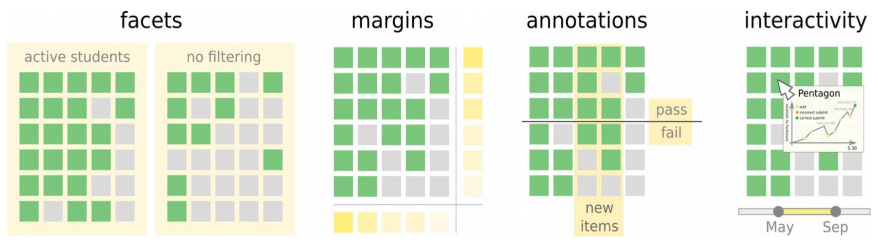

Figure 5. Four examples of student-item interaction matrix extensions.

3.3.1 Facets

Comparing multiple student-item matrices might bring a deeper insight than looking at just one matrix. We can compare sets of students (e.g., control vs. treatment group), sets of items (e.g., code comprehension vs. code writing), or time periods (e.g., June vs. November). In some cases, the comparison can be performed within a single matrix, using either groups or stacked cells (e.g., to compare predictions of multiple student models). If a single matrix is not sufficient, we can always arrange multiple matrices into a *facet grid*.

3.3.2 Margins

Any relevant student/item attributes can be added to the margins of the matrix. These attributes can be summaries of the displayed values in the rows and columns, facilitating the exploration of multiple levels of abstraction. Various summary curves can be seen as a projection of a specific student-item matrix. For example, a *survival curve* (Eagle & Barnes, 2014) is a projection that counts the number of interactions in columns of a time-ordered matrix, and an *item ordering bias curve* (Čechák & Pelánek, 2019) is a projection that averages columns in a time-ordered matrix with values set to the item presentation order. Figures 7 and 8 show these two summary curves represented as *heatlines*, which is a compact alternative to point plots or bar charts.

3.3.3 Annotations

When the matrix is used to deliver a message (e.g., in a research paper), we may want to highlight or delineate some parts. Examples of such annotations include a line showing a homework deadline, background highlight in columns of new items, and icons with exclamation marks put on the cells corresponding to interactions where a student model made a huge error in the predicted performance.

3.3.4 Interactivity

Interactive features can greatly simplify exploration on multiple levels of abstractions, readily providing details on demand (e.g., hovering over or clicking on a cell). Dropdowns and sliders can allow to easily select different subsets of data and change all the parameters of the student-item matrix discussed in previous sections. A valuable feature for debugging student models would be an option to interactively change the data, such as the observed performance of a student, to see how it would impact the model behavior.

4. CASE STUDY

In this section, we show several examples of student-item matrices using real-world data from an online learning system for learning programming. For this case study, we selected a single high-school class (28 students) and five item sets from a Python programming exercise (51 items in total). Each item asks students to write a short function, e.g., to compute a factorial or to detect a palindrome. In contrast to multiple-choice questions, these programming items take much longer to solve and most attempts are eventually successful. Also, the binary success is not the only relevant aspect of performance: the speed of the students and the quality of the code matter as well. Effenberger & Pelánek (2021) showed that considering these other aspects of performance is necessary for valid and reliable student models in this context.

4.1 Standard student-item matrix

Figure 6 shows a standard student-item matrix for this data. Cell colors represent the performance of a student on an item, considering both the product quality (code length) and fluency (speed of the student). The thresholds for each performance category are computed per item, using the length of the author's solution and data from all students (not just the single selected class). Unsuccessful interactions that took less than one minute are labeled as *not serious*. Effenberger & Pelánek (2021) demonstrated that for this programming exercise, such performance measure leads to valid and reliable estimates of skills and difficulties, while binary correctness does not. Computing the mean performance — which is shown in the margins — requires specifying a mapping from the discrete performance categories to numbers.

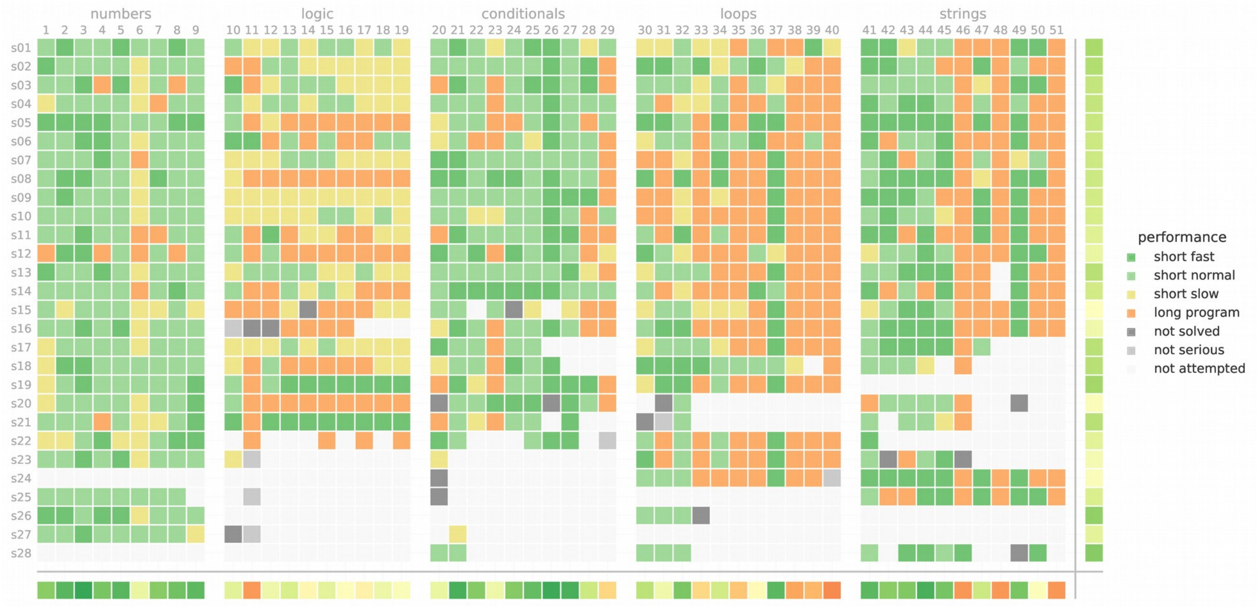

Figure 6. Standard student-item matrix for one class of students and a programming exercise with 51 items. Students are ordered by the number of attempted items. Items are grouped by item sets and ordered as in the learning system. The color of each cell represents performance and margins show mean performance for each student and item.

To the teacher, this visualization confirms that the selected exercise was a rather good fit for the class: neither were the items trivial for the students nor is any student extremely struggling. There are not huge differences in skills — the mean performance of all students is similar — but a few students did not even try to solve most of the items, indicating a lack of motivation.

Some topics are worth further practicing with the whole class: logic expressions, loops, and strings. A quick glance at the bottom margin suggests specific items that the teacher can analyze with the whole class (e.g., item 11 from the logic item set). In the last two item sets, the students managed to solve the items, just with a too long code, so the teacher can prepare an activity to specifically address this shortcoming, e.g., letting the students find a shorter solution to one of these items.

The same visualization would provide different insights to the developers and content authors, although they should select a larger and random sample of students to get a more representative picture. Had this been a random sample, we would conclude that some items are too difficult (e.g., item 11). We then could either make these items easier (e.g., by adding a hint or scaffolding) or make the students who encounter them more skillful (e.g., by moving the item at the end of the item set or adding similar but easier items to pretrain students).

Observing the frequency of too long programs, we might want to help the students to write shorter solutions. First, we should dig deeper and find what causes the excessive length. For example, in the logic item sets, many students fail to use "return (logic expression)" idiom instead of a four-line if-else block. With this knowledge, we can think of many possible interventions: adding scaffolded items demonstrating the use of this idiom, adding code refactoring items, showing a targeted feedback message to students who fail to use this idiom, and possibly even enforcing usage of this idiom by item structure, e.g., limiting the length of the program or the available code structures.

Yet different insights would this visualization provide to researchers who would like to use this data for student modeling. There are only a few unsolved attempts, rendering most of the current student modeling techniques useless since they focus on predicting the correctness of answers (Pelánek, 2017). A useful student model would need to consider also the other aspects of performance. Looking at the student-item matrix, we can guess which information is necessary for any student model to perform well. In our case, there is more variability across items than across students, and the item-average model seems like a reasonable baseline.

4.2 Time-ordered student-item matrix

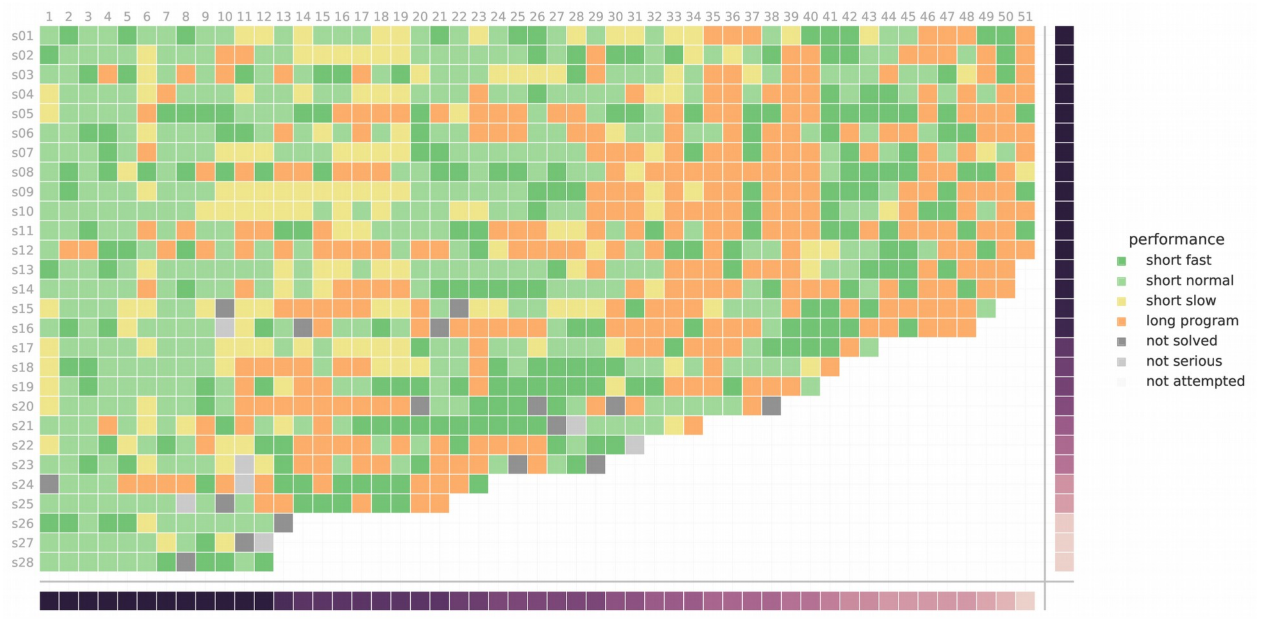

Figure 7. Time-ordered student-item matrix for the same class and items. Each cell represents one student-item interaction and its color denotes the student's performance. The columns correspond to the within-student order of the interactions. Margins show the total number of interactions.

For student modeling, the information about the order in which the students solved the items is crucial. In the time-ordered student-item matrix (Figures 7, 8), we can see which information is already available to the student model — just the previous cells on the same row — which can help us to understand its performance. Figure 7 shows the information used by a student model that disregards items and uses just the series of previous performances. While using only binary success for adaptation would be hopeless, there are streaks of the same performance category. We could use stacked or nested cells (discussed in section 3.1) to simultaneously show the difficulty of the attempted item or the prediction of a specific student model that we debug.

Figure 7 also reveals that the data are skewed, i.e., the number of interactions differ considerably between students; the bottom margin corresponds to the *survival curve*. Figure 8 use the same ordering but different aspect of the interactions: the presentation order of the item in the learning system. We can clearly see strong item ordering bias (Čechák & Pelánek, 2019), i.e., a high correlation between the presentation ordering and the order in which the students solve the items.

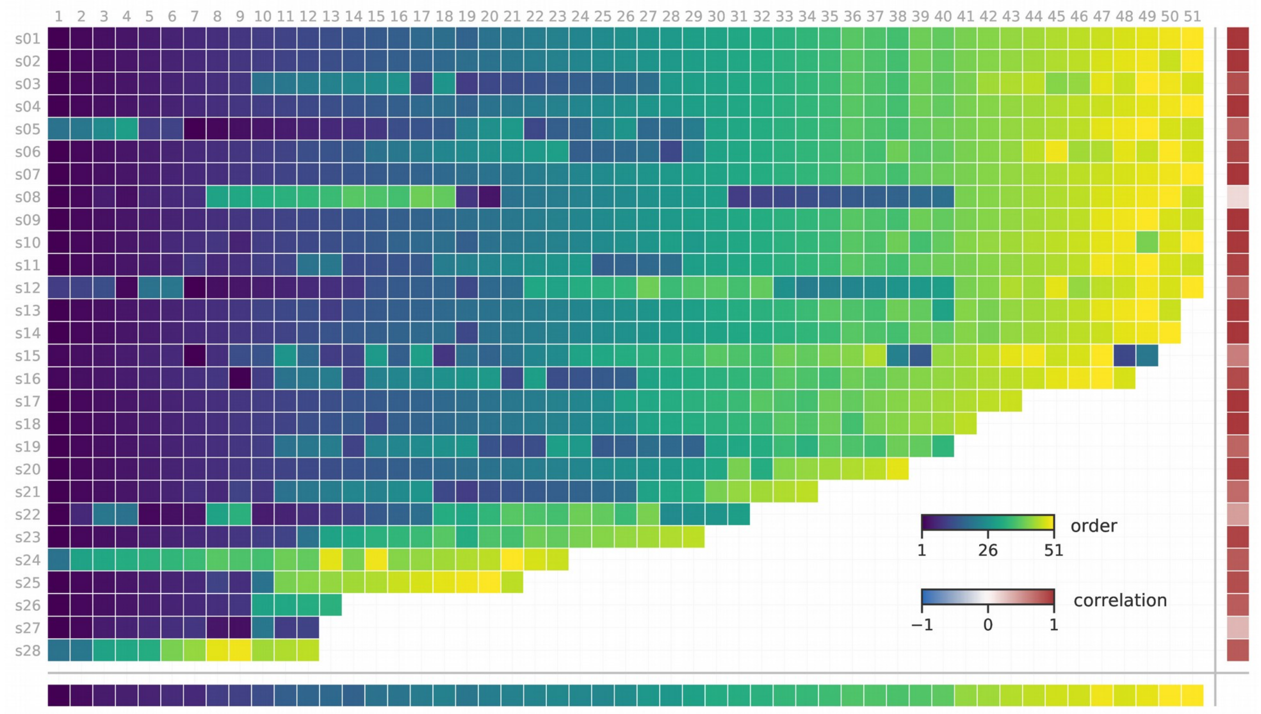

Figure 8. Time-ordered student-item matrix illustrating ordering bias. Each cell represents one student-item interaction; its column corresponds to the within-student order of the interaction, while the color corresponds to the presentation order of the item. The bottom margin shows the mean presentation order and the right margin shows the correlation between the presentation order and the within-student order.

4.3 Absolute-time student-item matrix

Instead of using the time just to order the interactions, we can directly place the interactions according to the specific date they happened. The resulting visualization is much less compact but gives a more complete picture of the student behavior over time. We can even see possible interactions between the students. Figure 9 illustrates such an *absolute-time* student-item matrix. Unlike the previous examples, here we use relative response time to color each interaction so that we can see suspicious streaks of fast solutions. It seems that there was a deadline on November 15 and many of the students might have been cheating. To confirm our suspicion, we looked at the submitted solutions of the suspicious students before the deadline. Indeed, many of these solutions were identical or unlikely similar.

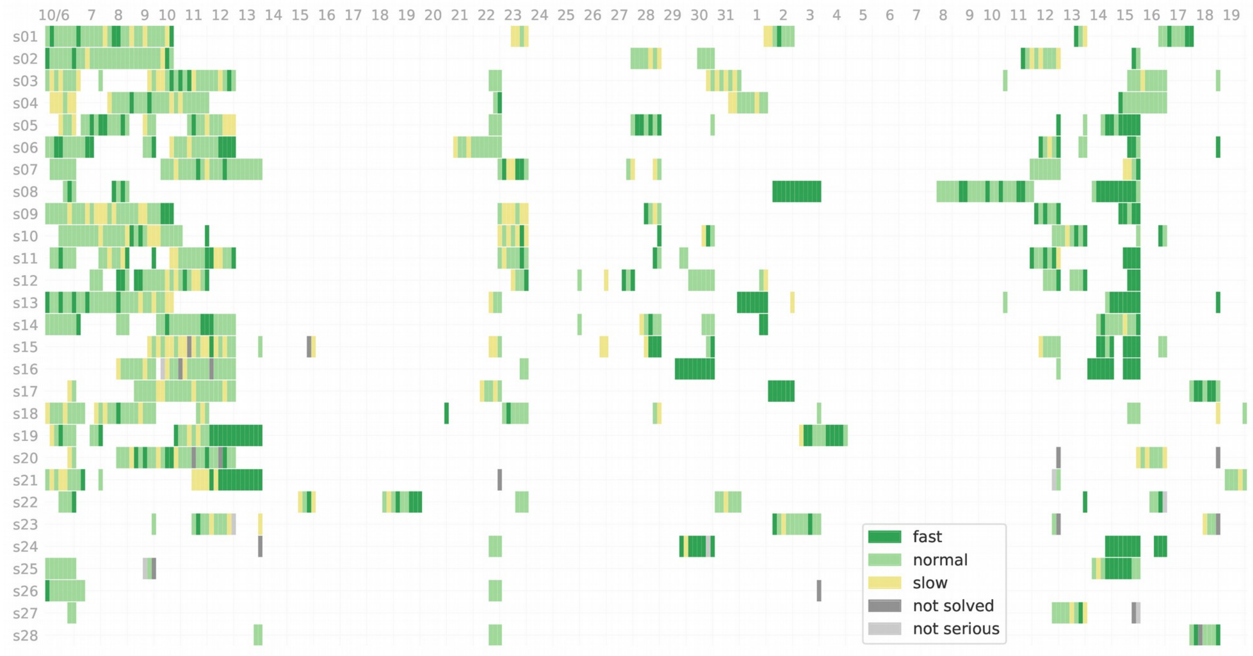

Figure 9. Absolute-time student-item matrix for the same class. Each cell represents one interaction; its horizontal position corresponds to the date, color to the relative speed of the student.

5. SUMMARY

In this chapter, we provided a detailed discussion of the visualization of the student-item interaction matrix, which is one of the approaches to visualizing interaction between students and educational items, focusing on a global overview of student activity ("multiple students, multiple items" view).

In practical applications of this visualization, it is important to start with the clarification of the purpose. The student-item interaction matrix can be visualized in many ways; the purpose of the visualization should guide the design choices. To facilitate these choices, we provided a systematic discussion of visualization aspects and design options.

The visualization is also useful for researchers. Before applying statistical or machine learning techniques, we recommend inspecting data using the visualization of student-item interactions. This visualization provides insight into the biases and peculiarities of the dataset. It can also be useful for understanding and explaining results.

There is also a potential for future research on the visualization itself. In Section 3.3, we outlined several extensions of the basic visualization approach; most of these deserve further attention and elaboration. Another important direction is the evaluation of visualization. The usefulness of a particular visualization depends on a particular use case and a dataset, so we cannot expect simple, universal evaluation results. However, the description of evaluation methods and specific case studies would certainly be useful.

REFERENCES

Aalst, W. M. van der, Guo, S., & Gorissen, P. (2013). Comparative process mining in education: An approach based on process cubes. In *International symposium on data-driven process discovery and analysis* (pp. 110–134). Springer.

Baker, R., Walonoski, J., Heffernan, N., Roll, I., Corbett, A., & Koedinger, K. (2008). Why students engage in “gaming the system” behavior in interactive learning environments. *Journal of Interactive Learning Research*, *19*(2), 185–224.

Bowers, A. J. (2010). Analyzing the longitudinal K-12 grading histories of entire cohorts of students: Grades, data driven decision making, dropping out and hierarchical cluster analysis. *Practical Assessment, Research, and Evaluation*, *15*(1), 7.

Brusilovsky, P., Somyürek, S., Guerra, J., Hosseini, R., Zadorozhny, V., & Durlach, P. J. (2016). Open social student modeling for personalized learning. *IEEE Transactions on Emerging Topics in Computing*, *4*(3), 450–461.

Čechák, J., & Pelánek, R. (2019). Item ordering biases in educational data. In *Proceedings of Artificial Intelligence in Education* (pp. 48–58). Springer.

Coffrin, C., Corrin, L., Barba, P. de, & Kennedy, G. (2014). Visualizing patterns of student engagement and performance in MOOCs. In *Proceedings of Learning Analytics and Knowledge* (pp. 83–92).

Confrey, J., Gianopulos, G., McGowan, W., Shah, M., & Belcher, M. (2017). Scaffolding learner-centered curricular coherence using learning maps and diagnostic assessments designed around mathematics learning trajectories. *ZDM*, *49*(5), 717–734.

Costagliola, G., Fuccella, V., Giordano, M., & Polese, G. (2008). Monitoring online tests through data visualization. *IEEE Transactions on Knowledge and Data Engineering*, *21*(6), 773–784.

Desmarais, M., & Lemieux, F. (2013). Clustering and visualizing study state sequences. In *Proceedings of Educational Data Mining*.

Eagle, M., & Barnes, T. (2014). Survival analysis on duration data in intelligent tutors. In *Proceedings of Intelligent Tutoring Systems* (pp. 178–187). Springer.

Effenberger, T., & Pelánek, R. (2021). Validity and Reliability of Student Models for Problem-Solving Activities. In *Proceedings of Learning Analytics and Knowledge* (pp. 1-11).

France, L., Heraud, J.-M., Marty, J.-C., Carron, T., & Heili, J. (2006). Monitoring virtual classroom: Visualization techniques to observe student activities in an e-learning system. In *Proceedings of Advanced Learning Technologies* (pp. 716–720). IEEE.

Fu, X., Shimada, A., Ogata, H., Taniguchi, Y., & Suehiro, D. (2017). Real-time learning analytics for C programming language courses. In Proceedings of the Learning Analytics & Knowledge (pp. 280-288).

Janssenswillen, G., Depaire, B., Swennen, M., Jans, M., & Vanhoof, K. (2019). BupaR: Enabling reproducible business process analysis. *Knowledge-Based Systems*, *163*, 927–930.

Johnson, M., Eagle, M., & Barnes, T. (2013). Invis: An interactive visualization tool for exploring interaction networks. In *Proceedings of Educational Data Mining*.

Khajah, M., Wing, R., Lindsey, R., & Mozer, M. (2014). Integrating latent-factor and knowledge-tracing models to predict individual differences in learning. In *Proceedings of Educational Data Mining*.

Lee, J. E., Recker, M., Bowers, A., & Yuan, M. (2016). Hierarchical cluster analysis heatmaps and pattern analysis: An approach for visualizing learning management system interaction data. In *Proceedings of Educational Data Mining* (pp. 603–604).

Mazza, R., & Dimitrova, V. (2007). CourseVis: A graphical student monitoring tool for supporting instructors in web-based distance courses. *International Journal of Human-Computer Studies*, *65*(2), 125–139.

Molenaar, I., & Knoop-van Campen, C. (2017). Teacher dashboards in practice: Usage and impact. In *European Conference on Technology Enhanced Learning* (pp. 125–138). Springer.

Monti, D., Rizzo, G., & Morisio, M. (2019). Visualizing ratings in recommender system datasets. In *IntRS@ RecSys* (pp. 60–64).

Nixon, T., Fancsali, S., & Ritter, S. (2013). The complex dynamics of aggregate learning curves. In *Proceedings of Education Data Mining* (pp. 338–339).

Northcutt, C. G., Ho, A. D., & Chuang, I. L. (2016). Detecting and preventing “multiple-account” cheating in massive open online courses. *Computers & Education*, *100*, 71–80.

Pelánek, R. (2017). Bayesian knowledge tracing, logistic models, and beyond: An overview of learner modeling techniques. *User Modeling and User-Adapted Interaction*, *27*(3-5), 313–350.

Pelánek, R. (2018). The details matter: Methodological nuances in the evaluation of student models. *User Modeling and User-Adapted Interaction*, *28*(3), 207–235.

Pelánek, R., & Effenberger, T. (2020). Beyond binary correctness: Classification of students’ answers in learning systems. *User Modeling and User-Adapted Interaction*, *27*(1), 89–118.

Reddy, S., Labutov, I., Banerjee, S., & Joachims, T. (2016). Unbounded human learning: Optimal scheduling for spaced repetition. In *Proceedings of Knowledge Discovery and Data Mining* (pp. 1815–1824).

Řihák, J., & Pelánek, R. (2016). Choosing a student model for a real world application. In *Building ITS bridges across frontiers (ITS workshop)*.

Sedrakyan, G., Snoeck, M., & De Weerdt, J. (2014). Process mining analysis of conceptual modeling behavior of novices–empirical study using JMermaid modeling and experimental logging environment. *Computers in Human Behavior*, *41*, 486–503.

Song, M., & Aalst, W. M. van der. (2007). Supporting process mining by showing events at a glance. In *Proceedings of Workshop on Information Technologies and Systems* (pp. 139–145).

Trcka, N., Pechenizkiy, M., & Aalst, W. van der. (2010). Process mining from educational data. *Handbook of Educational Data Mining*, 123–142.

Wallner, G., & Kriglstein, S. (2013). Visualization-based analysis of gameplay data–a review of literature. *Entertainment Computing*, *4*(3), 143–155.

Wang, C.-h., & Chen, C.-p. (2013). Employing online SP diagnostic table for qualitative comments on test results. *Electronic Journal of E-Learning*, *11*(3), 263–271.

AUTHOR INFORMATION

Complete name: Tomáš Effenberger

Institutional affiliation: Masaryk University, Faculty of Informatics

Institutional address: Botanická 68A, 602 00, Czech Republic

Complete mailing address: Botanická 68A, 602 00, Czech Republic

Telephone number: +420 737175251

Email address: tomas.effenberger@mail.muni.cz

Short biographical sketch (200 words maximum):
Tomáš Effenberger is a Ph.D. candidate in Computer Science at Masaryk University where he also received his master's degree. After research internships at the University of Oxford and Google AI, he became a member of the Adaptive Learning group at Masaryk University. His mission is to make learning more efficient and engaging by advancing techniques for improving adaptive learning environments.

Complete name: Radek Pelánek

Institutional affiliation: Masaryk University, Faculty of Informatics

Institutional address: Botanická 68A, 602 00, Czech Republic

Complete mailing address: Botanická 68A, 602 00, Czech Republic

Telephone number: +420 549496991

Email address: pelanek@fi.muni.cz

Short biographical sketch (200 words maximum):
Radek Pelánek received his Ph.D. degree in Computer Science from Masaryk University for his work on formal verification. Since 2010 his research interests focus on areas of educational data mining and learning analytics. Currently, he is the leader of the Adaptive Learning group at Masaryk University and is interested in both theoretical research in user modeling and practical development of adaptive learning environments.