

InVis: An EDM Tool For Graphical Rendering And Analysis Of Student Interaction Data

Vinay Sheshadri
North Carolina State
University
Raleigh, NC
vshesha@ncsu.edu

Collin Lynch
North Carolina State
University
Raleigh, NC
collin@pitt.edu

Dr. Tiffany Barnes
North Carolina State
University
Raleigh, NC
tmbarnes@ncsu.edu

ABSTRACT

InVis is a novel visualization tool that was developed to explore, navigate and catalog student interaction data. InVis processes datasets collected from interactive educational systems such as intelligent tutoring systems and homework helpers and visualizes the student data as graphs. This visual representation of data provides an interactive environment with additional insights into the dataset and thus enhances our understanding of students' learning activities. Here, we demonstrate the issues encountered during the analysis of large EDM data sets, the progressive features offered by the InVis tool in order to address these issues and finally establish the effectiveness of the tool with suitable examples.

Keywords

EDM, visualization, graphs, student interaction data

1. INTRODUCTION

One of the central goals of Educational Data Mining (EDM) is to translate raw student data into useful pedagogical insights. That is, educational dataminers seek to analyze student interaction data such as user-system logs with the goal of identifying: common errors, typical solutions and key conceptual challenges among other things. This research is of interest to learners, educators, administrators and researchers [17]. In recent years, the increased adoption of web-based tutoring systems, learning management tools and other interactive systems has resulted in an exponential increase in available data and increased demand for novel analytical tools. The Pittsburgh Science of Learning Center's DataShop, for example, currently stores over 188 datasets, encompassing 42 million student actions and 150,000 student hours [19]. With the increase in available data has come a corresponding increase in the insights EDM can provide and in making analytical tools available to expert instructors.

EDM researchers have generally relied on statistical analyses (see [14, 2, 1], formal rule induction (e.g. [12]), or other modeling methods to extract these insights. While these analytical methods are robust and have led to great progress in model development and evaluation, the increased interest in EDM by non-statisticians and practitioners has accentuated the need for "good visualization facilities to make their results meaningful to educators and e-learning designers" [16].

InVis was initially developed by Johnson, Eagle and Barnes [11]. The present version has been expanded to include changes to the visual editing system, export functions and other features. An example graph is shown in Figure 1. The graphical structure of InVis is designed to facilitate direct exploration of student datasets and easy comparison of individual solution paths. InVis can render individual student solutions or display the work of an entire class thus enabling educators to identify and draw insights from common student strategies and repeated mistakes [11]. InVis was inspired by the work of Barnes and Stamper [3] on the use of graphical representations for logic problems. Similar work has been done by Chiritoiu, Mihaescu and Burdescu who developed the EDM Visualization tool. This tool generates the student clustering models using k-means clustering algorithm [5]. However unlike InVis, the resulting visualization is non-interactive and non-graphical.

EDM researchers generally seek to answer questions such as: What actions can predict student success? Which strategy or solution path is more or less efficient and educationally effective? What decisions indicate student progress? And what are the features of a learning environment that promote learning? (see [15]). In a programming tutor, for example, students might be given the task of implementing an array-sorting algorithm for a large vector of integers. The particular choice of algorithm and the implementation details are left to the students to formulate using a variety of existing tools. This resulting code will proceed in several stages including reading data from disk, sorting the contents in memory, and returning the result. Our goal as researchers is to classify the successful students, identify the most commonly-chosen algorithms and flag individuals who faced difficulties or failed to complete the assignment. In a logic tutor such as Deep Thought [7] or a Physics tutor such as Andes [20] we would like to make similar determinations by focusing on the solutions chosen by the students and the individually-critical steps.

The graph representation provided by InVis allows us to answer these questions by constructing and exploring interactive visualizations of the student dataset. By rendering a graph of a class or key subgroup (e.g. low-performing students), we can visually identify garden-path solutions over long isolated chains, identify critical states through which most students traversed and so on. These visualizations can also be used to guide, or evaluate the output of automatic analysis such as MDP models or path-detection algorithms. In the remainder of this paper we will discuss the tool, describe key features of it in detail and illustrate the type of insights it can provide.

2. DATA

We will illustrate the operation of InVis on a typical dataset. For the purposes of the present paper we will use student data collected from the Deep Thought tutor [6, 7]. Deep Thought is a graph-based tutor for first-order logic. Students using the system are presented with a problem defined by a set of given components (e.g. " $A \wedge \neg B \wedge C \Rightarrow B$ ") and are tasked with proving some goal state (e.g. $\neg C$). Problem solving proceeds through forward or backward-chaining with students applying rules such as Modus Ponens or Modus Tolens to draw new conclusions. For example, given the conclusion B , the student could propose that B was derived using Modus Ponens (MP) on two new, unjustified propositions: $A \rightarrow B, A$. This is like a conditional proof in that, if the student can justify $A \rightarrow B$ and A , then the proof is complete. At any time, the student can work backwards from any unjustified components, or forwards from any derived statements or the premises [8].

The DT data thus has a number of key characteristics that make it amenable to graphical display. The data is grouped into fixed problems covered by many students. Each problem is defined by a static set of given information and a clear goal. And the solutions are constructed via iterative rule applications drawn from a fixed library. As a consequence it is possible to define a fixed, albeit large, space of solution states and to efficiently map the traversal between them. While this seems restrictive this set of criteria applies to data collected from many if not most Intelligent Tutoring Systems. Andes, for example, defines problems by a set of given values (e.g. " $M_{car} = 2kg$ ") sets fixed variable goals (e.g. " S_{car-t_0} ": speed of the car at t_0) and groups student actions into a fixed set of rule applications. Similar state representations have also been applied to other datasets such as code-states in the SNAP programming tutor [4].

The figures shown below are drawn from two InVis datasets. We will focus in detail on a small dataset comparing the work of three students on a single problem with a fixed set of givens and two alternate goals. Such a small dataset is designed to allow for efficient illustration but is not an upper limit for analysis. We will also present some qualitative discussion of larger scale analysis with a larger DT dataset as shown in Figure 3.

3. FEATURES OF INVIS

InVis was developed with the *Java Netbeans Framework* and employs the *JUNG* libraries for the rendering of the graphs [13]. It provides an assortment of features that allow the end user to interact with the visualizations and draw obser-

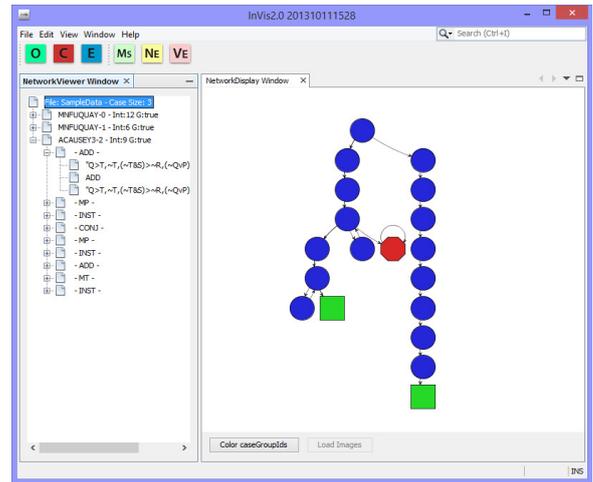


Figure 1: Network Display and Viewer

vations from the data set. The *Network Display*, *Network Viewer*, *Visual Editor* and *Export Dot Data* are some of the prominent features of InVis which will be illustrated with examples in the upcoming sections. InVis also supports *MDP calculation*, *between-ness calculation* and *frequency reduction* which currently are under development and test phases.

3.1 Network Display and Viewer

The front-end of InVis is the *The Network Display* component. It displays the interaction network generated by the engine in a graphical format. The user is presented with a cumulative overview of the processed input data. The various logic states of the DT tutor are represented by nodes and the applied propositional logic transformations are represented by edges of the graph. Intermediate states are represented by blue circular nodes while the goal states are represented by green square nodes. Error states in the DT dataset are defined by logical fallacies and are represented by red octagons for easy identification. The sample display shown in Figure 1 contains 16 intermediate nodes arrayed from the top to bottom of the network, one error state located in the center, and two goal states at the bottom.

The *Network Viewer* component represents the InVis input data in the form of a tree structure known as case-set. Each primary node in the case-set represents a student and each sub-node under it represents a transition state executed by the student sequentially. Selecting a student in the Network Viewer window highlights the corresponding path in the Network Display window. Selecting a sub-node highlights the corresponding nodes and edges that were involved in the transformation. Expanding a sub-node will cause the system to display the pre-state and post-state information from the nodes involved in that transition.

The path taken by a student to solve the given problem can be detected by selecting the appropriate student in the Network Viewer window. This will fade the non-path nodes to bring the chosen path to the foreground. An example of this highlighting is shown in Figure 2 where we have selected a single student path within the demo dataset.

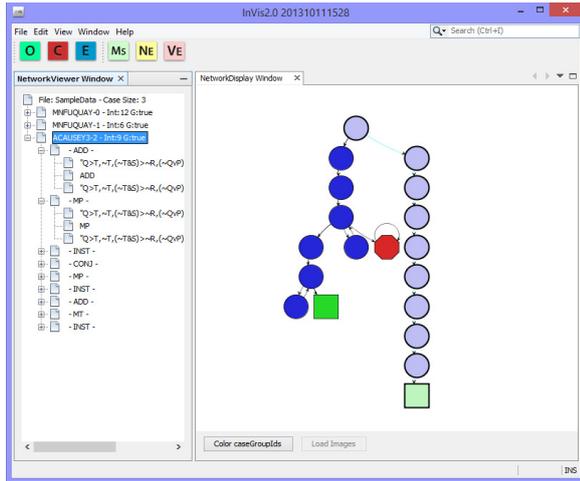


Figure 2: Tracing the path of a student

One common use of InVis is to identify frequently-occurring error states. The system can also be used to analyze the different paths taken by students in order to achieve a common goal and isolate the areas where the students face difficulties in solving the given problem or took a garden path. A garden path is an inefficient path from one target state to another with many nonessential intermediate states. From Figure 1, in the current data set, for example, one student performed 11 transitions to achieve the goal, due in part to cycles, whereas a separate student reached the goal with 5 transitions. Each transition is marked by an arc from one state to another in the graph. Thus the Network Display provides an instructor with a cumulative analysis of the input data and aids the instructor in identifying areas of difficulty faced by students during the course of problem solving.

Figure 3 shows the visualization generated by InVis for a sample large dataset. The bold edges indicate the common paths employed by the students in order to solve a given problem. The graph also highlights the garden paths and the succeeding action taken by students towards achieving the goal states. From the rendered visualization it is clear that the cloud space comprises of students who achieved the goal, indicated in green and students who failed to reach the final goal states. InVis can thus be employed to congregate useful observations on large EDM datasets.

3.2 Visual Editor

The *Visual Editor* component of InVis controls the various visual aspects of the graph displayed in the Network Display window. The visual editor provides options for displaying the node and edge data of the graph. InVis renders graphs with the DAG tree layout as the default layout. The visual editor provides options for rendering the graph in different layouts. An ISOM layout of the originally generated graph is shown in Figure 4.

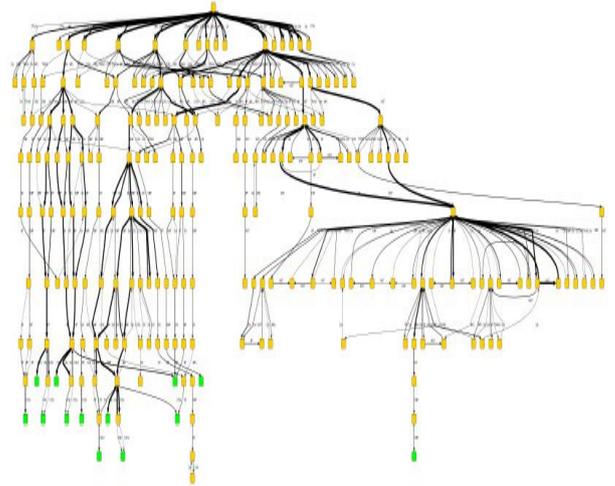


Figure 3: InVis and large data sets

The Visual Editor also provides an option for normalizing the edge widths based on the case frequencies. Case frequencies are defined by the number of students who used the same transition between the given set of states. When the *Normalize Width* option is selected, InVis reloads the graph with width of edges proportional to the case frequency. This feature helps instructors in identifying the logic states and transitions which are most used by the students.

The Visual Editor can be launched by clicking on the Visual Editor icon in the toolbar. Options are provided in the Visual Editor window to control the display of node and edge labels. A notable option provided by the visual editor is the option to normalize edge widths. Normalizing edge widths results in the modification of the edge widths of the graph in proportion to the case frequencies.

Figure 5 displays the zoomed in version of the graph with normalized edges. Edges with case frequency of 2 have thicker connecting lines compared to the edges with case frequency of 1. Thus the thickness of the edge offers a visual cue to the instructor in identifying the most commonly traversed paths by students when achieving the given goal.

3.3 Exporting InVis Data

Graphviz is a heterogeneous collection of graph drawing tools [9]. The software is available under open source license. The input to the Graphviz tool is a description of the required graph in a simple text language such as DOT. The tool processes the input and renders output graphs in useful formats, such as images and SVG for web pages; PDF or Postscript for inclusion in other documents; or display in an interactive graph browser [10]. Graphviz has many useful features for concrete diagrams, options for colors, fonts, tabular node layouts, line styles, hyperlinks, and custom shapes.

In order to leverage the graph design features offered by Graphviz, InVis now features a new export option which

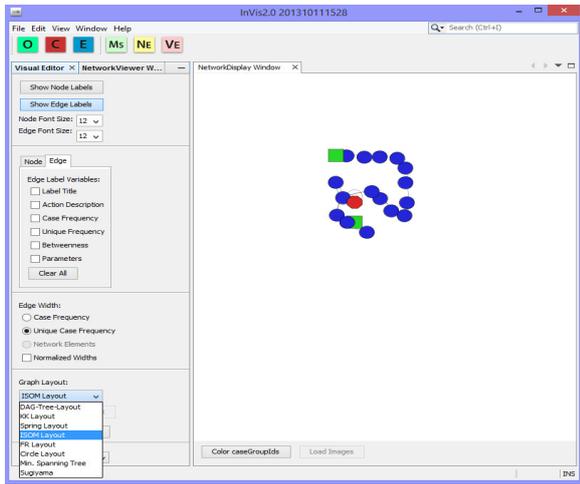


Figure 4: Different graph layouts

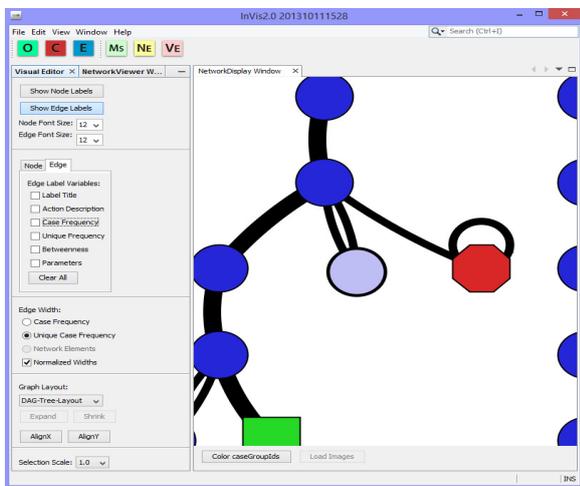


Figure 5: Normalized width - Zoomed in

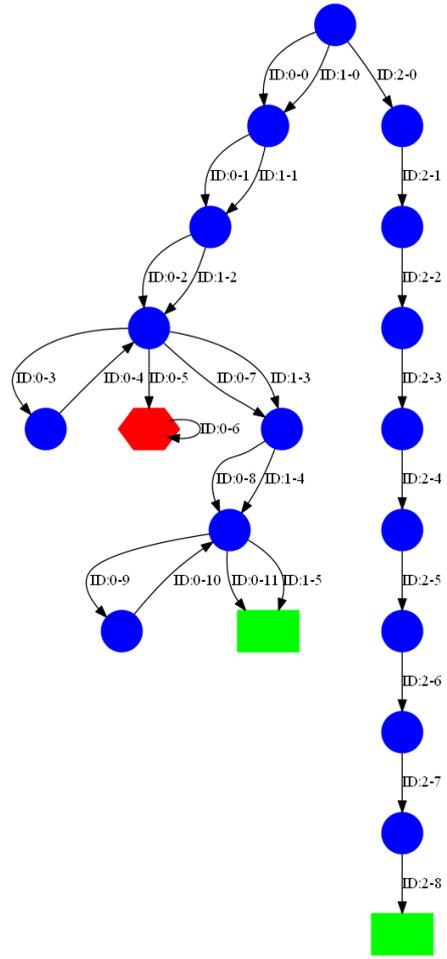


Figure 6: Exported data loaded in Graphviz

renders the input Deep thought data into a DOT format file. The DOT file can be directly imported by Graphviz to generate static images such as PNG, JPEG or interactive formats such as SVG. These visualizations will match those generated by the Network Display tool. Figure 6 shows a graph generated by Graphviz using exported InVis data. Here the arcs are annotated via a static ID number that helps in manually identifying the states and transition information. This data is captured as part of the export process.

4. DISCUSSION

The graphical rendering of EDM data via InVis can yield unique insights into the student interaction data. Romero and Ventura classified EDM objectives depending on the viewpoint of the final user as learner, educator, administrator and researcher [17]. InVis supports learners by providing visual feedback and recommendations to improve performance. Students can compare their approach with that of other students graphically. This can promote real time self-assessment and adoption of better approaches to problem solving.

Educators can use the tool to identify good and poor student solutions and to better understand the students' learning processes which can, in turn, reflect on their own teaching methods. The graphical summary presented by InVis gives an overview, and allows for detailed exploration of, the paths taken by students in achieving a solution to a given problem.

The presence of garden paths, loops and error states illustrate areas where the students have encountered difficulties in deriving a solution to a given problem. This empowers researchers with visual data to model suitable hint generation techniques that can deploy automatic corrective actions [18]. InVis can assist administrators to reorganize institutional resources based on visual evaluation of the effectiveness of a teaching method adopted in a particular course.

In the case of the sorting example introduced in the earlier section, by normalizing the edge width, we can identify the most commonly used sorting algorithm. We can also identify the optimal solution to the given problem comparing the number of transition states between the start and end goal

