Visualization 2 - Submission2

Kehlbeck et al. 2022 - SPEULER: Semantics-preserving Euler Diagrams

Maria Mußner - 51870828
Ana Jankovic - 12131843

1. Data Preparation

The data is loaded from the following link:
(https://raw.githubusercontent.com/sharmaroshan/Zoo-Dataset/master/zoo.csv).
We chose this data set, because it has highly overlapping data and contains boolean values, which are necessary for the approach of our chosen paper.
The *load_and_process_data()* function loads the data, removes invalid instances (such as instance of girl) and converts numerical attributes to one-hot-encoded labels. Finally, the data is converted to integer values of zeros and ones.

2. Generating graphs

In order to generate the graphs, we first need to select data instances and attributes on the GUI. To confirm the selection the *'update'*-Button has to be pressed to process the data.
Firstly, the existing intersections of attributes (sets) based on the selected data are calculated. If there exist no intersections, the appropriate text information is written. In case we have some intersections, the function *createGraphs()* is called.

createGraphs() checks if we have subsequent ranks in the intersection (e.g. rank 1 with the intersections: "a" and "b", and the rank 2 with the intersections: "a/b"). If a rank is not represented in the set of intersections (e.g. we have just rank 1 with "a", "b" and "c", and rank 3 with "a/b/c"), we visualize only the abstract graph. If we have all subsequent ranks, we visualize the other graphs, starting with an abstract graph using the *visualizeAbstract()* function. Also, we calculate the order of the nodes of each rank using the function *odered_rank_nodes()* (see 2.2), and we create adjacency matrices for each rank.
The adjacency matrix for one rank is the matrix, whose rows contain nodes of rank r, and its columns correspond to the nodes of rank r-1. If node on rank r is the child of the node on rank r-1 (e.g. "a/b" is a child of "a" and "b" because it extends them) the matrix will be filled with 1 in the corresponding entries.
Further, we convert the adjacency matrices to masks containing true or false values using the createMatrix*()* function (see 2.2). Based on these masks, we visualize the Euler dual graph in *visualizeEuler().* Furthermore, for visualizing the curve graph we calculate, for each node if it has at least one parent and one child. In case it does, we proceed with visualizing final curves with *visualizeCurves().* Additionally to the paper we add the names of the instances on the top of the final

curves. In some irregular cases that cannot be visualized, it is possible to have a curve that is actually equal to one point, and in such cases the instances cannot be added.

Further, we create a color scale based on the selected attributes, assigning one color to each of the chosen attributes.

2.1. Abstract Graph

The abstract graph is the fundamental step to visualize the graph. Here we draw a circle for each set of intersections in our selected data, where each row corresponds to the set of intersections of the according rank. We implemented this in the following way:
In *visualize_abstract()* we organize the nodes in rows, and scale them to an according size on the webpage. For each element in the group we draw a circle with the according text of the attributes to the node. The result shows the data organized according to the rank of the intersection sets.

2.2. Euler dual graph

For proper visualization of euler dual graph, we first need to order the nodes of each rank using the ordered_rank_nodes*()* function within *createGraphs()* to avoid intersections of lines. The first rank is sorted in such a way, that the first node contains the smallest number of occurrences in the intersections (e.g. if we have “a” and “a/b” intersections, “a” has 2 occurrences and “b” only one so it will be first). The other ranks are sorted, such that they lead to the longest sequence of consecutive ones in each of the adjacency matrices.
The first element of the adjacency matrix needs to be 1. For the first row, it’s possible to have 1 at the last position of the row as well. The last node should have 1 at the last position of the adjacency matrix, and it is allowed to have 1 at the first position. The other rows are filled with ones to lead to the longest consecutive ones sequences across an adjacency matrix.
The *createMatrix()* function masks all ones in the adjacency matrix, that are apart from the sequence of consecutive ones with 0. This way we avoid overlapping lines, when connecting the nodes in the Euler dual graph.
visualizeEuler() draws the nodes in the calculated order. We draw lines according to the computed masks with a color coding, such that the color of the lines is of the color coding of the extending attribute e.g. for the line between “a/b” and “a”, the line will have the color according to “b”.

2.3. Circular layout

visualizeCircular() places the nodes from the Euler dual graph in a circular layout. The node with the highest rank r is placed in the center. The other nodes are placed on the surrounded rings based on their rank. The nodes from the rank r-1 will be distributed evenly on the next ring. The nodes from the ranks above are placed on the rings, such that if a node on rank r-2 has two child nodes on rank r-1, it will be placed with the mean angle of the child nodes.
If it has only one child node, the angle of this node will be the same as the angle of the child node. If it has 3 children, it will be placed with the angle of the middle child node. The lines are generated in the same way as in the Euler dual graph.

In this function we also calculate the control points that are used for creating curves of sets. When creating the lines, we find the mean value of them and the position is one control point. We also keep track of the color of this point (because only this line will be able to pass through this point). The other type of control points, the gates, are placed on the rings between the nodes. They are placed in the middle of each node pair.
The *getGatesForColors()* function assigns gates to each color. A line of a certain color (certain attribute curve) can pass only through those gates, when passing from one rank to another.
For each gate we do the following: if the surrounding nodes of this gate have the common parent, we assign it to this gate of the color of the lines that are connecting the parent with both nodes (i.e. just the lines with these two colors will be able to pass through this gate). A similar procedure is done, if they have a common child. If they do not have a common parent or common child, we are looking for the common grandchild. All the subsequent children of each node must be adjacent. The gate will have the colors of all the lines that are connecting children with their children.

2.4. Final graph - drawing curves

createCurves() is finally visualizing the curves that represent sets for each attribute. The control points are firstly ordered to be connected. This is done with the function *getOrderedControlPoints()* (see below). After that, we visualize the curves using catMull-Rom splines.

getOrderedControlPoints() takes all the control points and the starting point. The first point is the one that is on the center of the most further line from the center point. Then we split the area to the left and to the right part relative to the line that connects the center of the graph and the first point. The next point is searched on the same side as the previous point. The next point is the closest one. If the previous point was the node, we consider the current direction. The next should be on the lower rank, if we are going in the direction from the higher to the lower rank, and vice versa. If we do not have any appropriate control point on the current side, we switch the sides until we reach the last control point. The last point corresponds to the first point.

2.5. Visualizing instances

The *visualizeInstanceNames()* function draws the text of the instances that we selected, in the corresponding area of intersection that it belongs to. The positions are randomly generated.

3. Working examples

This approach has some limitations. We can visualize the subsets of data only if our selection has a sink node for all the nodes, none of the ranks is skipped and each node has at least one parent and at least one child. Therefore, we provided some instances that are working:

1. attributes : eggs, milk
   instances : e.g. all the instances (select the first one, hold shift and select the last one)
2. attributes : feathers, eggs, aquatic, predator, catsize
   instances : aardvark, bear, chicken, crab, crow, dolphin, dove, duck, flamingo, gorilla, goat, gull, kiwi, ladybird, lark, moth, octopus, opossum, ostrich, penguin, piranha, platypus, pony, raccoon, scorpion, seal, seasnake, slug, swan, slowworm, termite, tortoise, vulture, wasp
3. attributes: same as above, without aquatic
   instances : same as above
4. attributes : feathers, eggs, aquatic, predator
   instances : same as above