GraphMDL Visualizer: Interactive Visualization of Graph Patterns

Francesco Bariatti, Peggy Cellier, Sébastien Ferré

GraphMDL Visualizer: Interactive Visualization of Graph Patterns

Francesco Bariatti, Peggy Cellier, and Sébastien Ferré

Univ Rennes, INSA, CNRS, IRISA
firstname.lastname@irisa.fr

Abstract. Pattern mining algorithms allow to extract structures from data to highlight interesting and useful knowledge. However, those approaches can only be truly helpful if the users can actually understand their outputs. Thus, visualization techniques play a great role in pattern mining, bridging the gap between the algorithms and the users.
In this demo paper we propose GraphMDL Visualizer, a tool for the *interactive visualization* of the graph patterns extracted with GraphMDL, a graph mining approach based on the MDL principle. GraphMDL Visualizer is structured according to the behavior and needs of users when they analyze GraphMDL results. The tool has different views, ranging from more general (distribution of pattern characteristics), to more specific (visualization of specific patterns). It is also highly interactive, allowing the users to customize the different views, and navigate between them, through simple mouse clicks. GraphMDL Visualizer is freely available online.

Keywords: Pattern Mining · Graph Mining · Minimum Description Length · Visualization

1 Introduction

Analyzing data can be a daunting task for humans, especially when the amount of data is important. *Pattern mining* algorithms —and in particular *graph mining algorithms* [10,7]— make the task easier by extracting structures that appear frequently in the data, allowing the users to concentrate only on those specific structures. The GraphMDL algorithm [4] further helps users by leveraging the Minimum Description Length principle [5,8] in order to select only a small subset of patterns that are deemed descriptive of the data. That way, the users are not overwhelmed by the large amount of patterns extracted by classic graph mining approaches. However, all these pattern mining approaches can only be truly helpful if the users can actually understand their outputs: what the extracted patterns look like, how they appear in the data, and alongside which other patterns. Thus, visualization techniques play a great role in pattern mining, bridging the gap between the algorithms and the users [6].

In this demo paper we propose GraphMDL Visualizer, a tool that presents the users with an *interactive visualization* of GraphMDL results. We construct

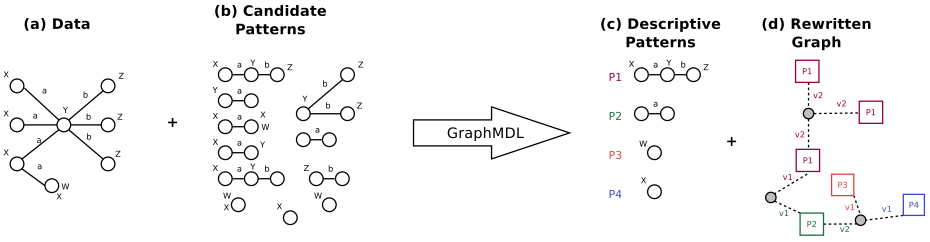


Fig. 1. The GraphMDL approach takes as input a data graph and some candidate patterns. It outputs a set of descriptive patterns and a "rewritten graph" showing how those patterns can be used to represent the data.

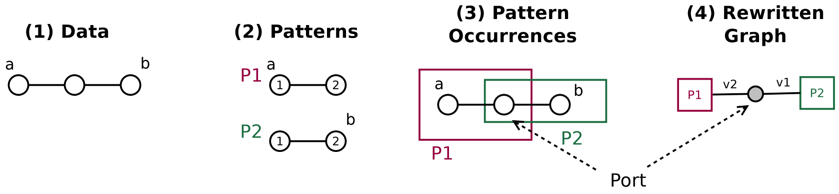


Fig. 2. GraphMDL creates a *rewritten graph* to describe the data as a composition of pattern occurrences. The vertices that are at the boundaries between pattern occurrences as called *ports*.

GraphMDL Visualizer around the needs and expected behavior of users when analyzing GraphMDL results. The tool has different views, ranging from more general (distribution of pattern characteristics), to more specific (visualization of a specific pattern). It is also highly interactive, allowing the users to navigate between the different views and customize them through simple mouse clicks. Since GraphMDL results are richer than other graph mining approaches (see the concepts of "usage" and "ports" described in Section 2), a custom tool is needed for visualizing them at their full potential. The different views of the tool can show all the information provided by GraphMDL in one single page, with an organization of the content that we hope is beneficial for the analysis. GraphMDL Visualizer is freely available online at `http://graphmdl-viz.irisa.fr/`.

The paper is organized as follows. Section 2 reminds the general behaviour of GraphMDL and some key notions that help understand the rest of the paper. Section 3 presents GraphMDL Visualizer in details, through a user story and some screenshots.

2 GraphMDL Background Knowledge

Pattern mining algorithms help users understand data by extracting patterns, i.e. structures, that appear (frequently) in the data. *Graph mining* algorithms do so when data and patterns are graphs. However, classic graph mining algorithms (e.g. gSpan [10] and Gaston [7]) tend to extract too many patterns for users to analyse, because of the large number of possible patterns. The GraphMDL approach [4] uses the *Minimum Description Length* (MDL) principle [5,8] to reduce the number of graph patterns extracted by selecting the most descriptive patterns among them. It was inspired by the Krimp algorithm [9], which does a similar task with itemset patterns on transactional data.

In this section we remind the main principles of GraphMDL useful to understand GraphMDL Visualizer. More details about GraphMDL are given in [4].

Fig. 1 summarizes the general behaviour of GraphMDL. The approach takes as input a *data graph* (a) and a set of *candidate patterns* (b) that have occurrences in this data graph. These patterns can be extracted with any classic graph mining algorithm (e.g. gSpan is used in [4]). It outputs a set of patterns it deemed as "descriptive" (c) and a *rewritten graph* (d), a structure that describes how the selected patterns can be inter-connected to represent the data graph.

In order to choose which set of patterns is the most descriptive, GraphMDL needs a way to evaluate each possible set to compare it with the others. For this, it uses the MDL principle [5,8]. This principle comes from the domain of information theory and states that the best model (out of all possible models) to describe some data, is the one that minimizes the *description length* of the data encoded by the model. Roughly, the description length of an element measures the quantity of information in the element. GraphMDL instantiates the MDL principle similarly to the Krimp algorithm [9], from which it is inspired: the data is "represented" (or *encoded*) as a composition of pattern occurrences[1]. This encoding is then given a *description length* that allows to evaluate the set of patterns.

The main contribution of GraphMDL is how to use graph patterns to cover graph data, and a MDL encoding that works in that context. Graphs are a complex type of data since they have a structural component. For example, in Fig. 2 knowing that patterns P1 and P2 are present in the data is not sufficient, it is important to know how they are *connected*: in this case the second vertex of P1 and the first vertex of P2 correspond to the same vertex in the data. In order to encode this structural component, GraphMDL introduces the notion of *ports*, which are vertices that are shared between multiple pattern occurrences (e.g. the vertices marked by an arrow in Fig. 2).

When covering the data with some patterns, GraphMDL produces what it calls a *rewritten graph*, which is a structure that states how pattern occurrences, connected through ports, form the data graph. This structure is used when computing a description length, and is also given to the users, which can use it for interpretation. For example Fig. 2 (4) shows a rewritten graph: from it we can learn that in the data there are an occurrence of P1 and one of P2 (squares in the image), and that these two occurrences share a vertex (in grey).

Lastly, every pattern has a *usage*: it corresponds to the number of times that the pattern is used in the data graph encoding i.e. the number of occurrences of this pattern in the rewritten graph. For example in Fig. 2 each pattern has a usage of 1 and in Fig. 1 pattern P1 has a usage of 3.

[1] It can also be said that the patterns *cover* the data

3 The GraphMDL Visualizer Tool

In order to design GraphMDL Visualizer we asked ourselves what would be a user's *story* when trying to understand GraphMDL output. We identified 4 steps in the interaction between the user and the algorithm's output, and designed GraphMDL Visualizer around them.

Step 1. When confronted for the first time with some new data and patterns, the user wants to start with a broad view of statistical information about the patterns: How many patterns have been extracted? Are the extracted patterns very frequent and/or big? (i.e. what is the distribution of the *usage*[2] and size of the patterns?).

Step 2. After that, the user wants to look more precisely at the individual patterns: for example, they may start by looking at the largest pattern extracted, or the most frequent: What are their defining characteristics?

Step 3. Then, the user wants to know what the pattern(s) they selected during the previous step look like. Graphs are complex structures that are usually best understood when plotted graphically.

Step 4. Finally, the user wants to know how the patterns are used to *cover* the data: Which are the neighbours of the pattern they selected at the previous step(s)? How a certain subset of the data is expressed in terms of pattern occurrences?

During step 4, the user identifies new patterns that pique their interest, and navigates to a different step, or starts over with a different goal in mind. We found that this back-and-forth navigation was very typical, and therefore a good visualization should be interactive and allow the user to easily move from one view to another.

3.1 Implementation

GraphMDL Visualizer is an interactive web page that the user can open in their web browser. Web applications have a native support for user interactivity (i.e. reacting when the user clicks or moves the mouse), and can be used without a complex installation process. The application has been developed using the Javascript framework VueJS [3], the Javascript data visualization library D3.js [2], and the CSS library Bootstrap [1]. After GraphMDL has been run[3], the user just needs to upload the produced output file to GraphMDL Visualizer.

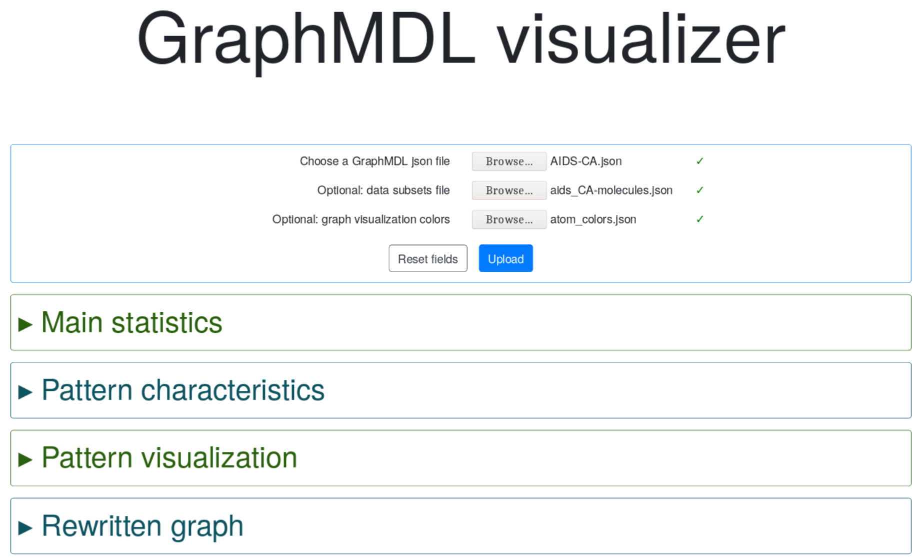
The source code for GraphMDL Visualizer is available as a git repository[4]. The application is also available online at `http://graphmdl-viz.irisa.fr/`.

Fig. 3 shows the general structure of GraphMDL Visualizer. The application contains 4 main blocks, which correspond to the 4 steps of the user story that we described at the beginning of this section. As a running example, we imagine

[2] In GraphMDL the usage is a specific notion: it represents how many times a pattern appears in the rewritten graph (see Section 2).

[3] We use the implementation at `https://gitlab.inria.fr/fbariatt/graphmdl/`

[4] `https://gitlab.inria.fr/fbariatt/graphmdl-visualizer`



GraphMDL visualizer

Choose a GraphMDL json file: AIDS-CA.json ✓

Optional: data subsets file: aids_CA-molecules.json ✓

Optional: graph visualization colors: atom_colors.json ✓

Fig. 3. The main structure of GraphMDL Visualizer. The application is composed of 4 blocks, which correspond to the 4 steps of data analysis that we identified in Section 3.

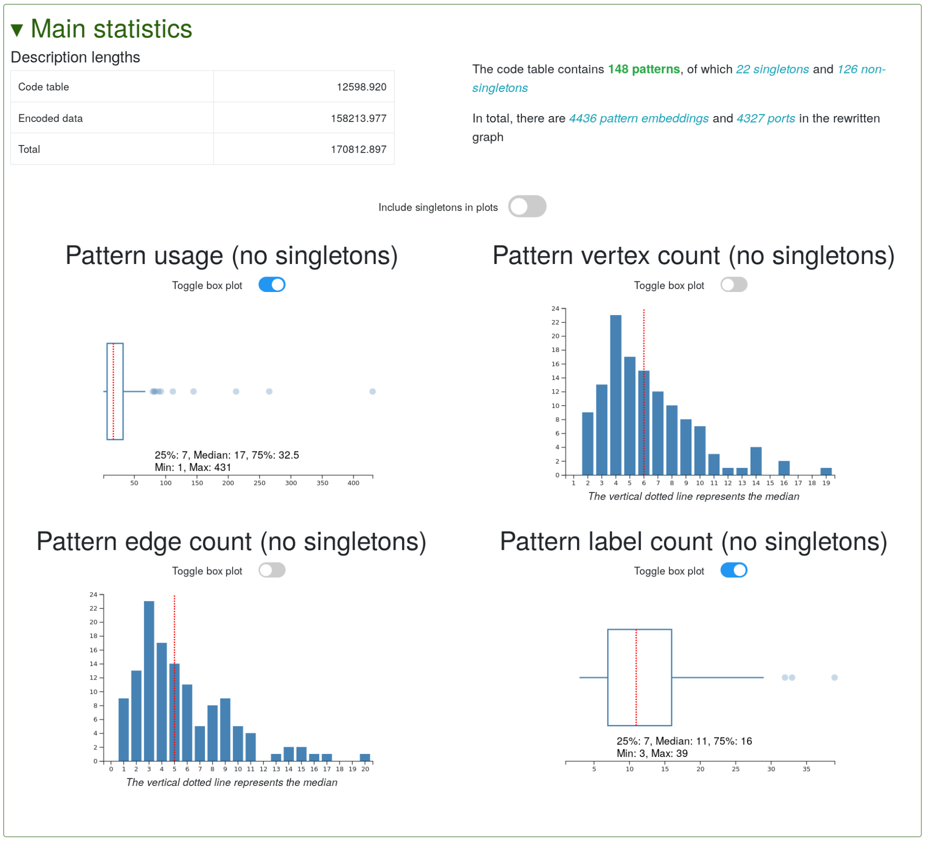
that the users ran GraphMDL on the molecular dataset AIDS-CA[5], which has been used in [4] to evaluate GraphMDL.

3.2 Step 1: Main statistics

When the user first starts analysing GraphMDL results, we expect them to want a general view of the selected patterns, without focusing on the specifics of each pattern (yet). The "Main statistics" block, shown in Fig. 4, provides this view. At the top-right, some general pieces of information are given, notably the number of patterns selected. At the top-left, description length information is reported: this information is related to the MDL approach and is mainly useful when comparing different pattern sets or datasets.

The bottom part of the block contains 4 plots. These plots represent the distribution of pattern usages and sizes (different size measures are present: vertex count, edge count, label count). A notable advantage of having an interactive web page w.r.t. classic static plots is that the users can zoom and drag those plots, to higlight the areas that interest them the most. Hovering a bar of the bar plot also shows the exact height of the bar. Each plot also has a button to toggle between a bar plot or a box plot for representing the data. In this way the user can switch between the two visualizations, depending on whether they want to observe the exact distribution or its main characteristics (median, percentiles). For example in Fig. 4, we can observe that 75% of the patterns have a usage

[5] https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data



▾ Main statistics

Description lengths

Code table	12598.920
Encoded data	158213.977
Total	170812.897

The code table contains **148 patterns**, of which *22 singletons* and *126 non-singletons*

In total, there are *4436 pattern embeddings* and *4327 ports* in the rewritten graph

Include singletons in plots ○

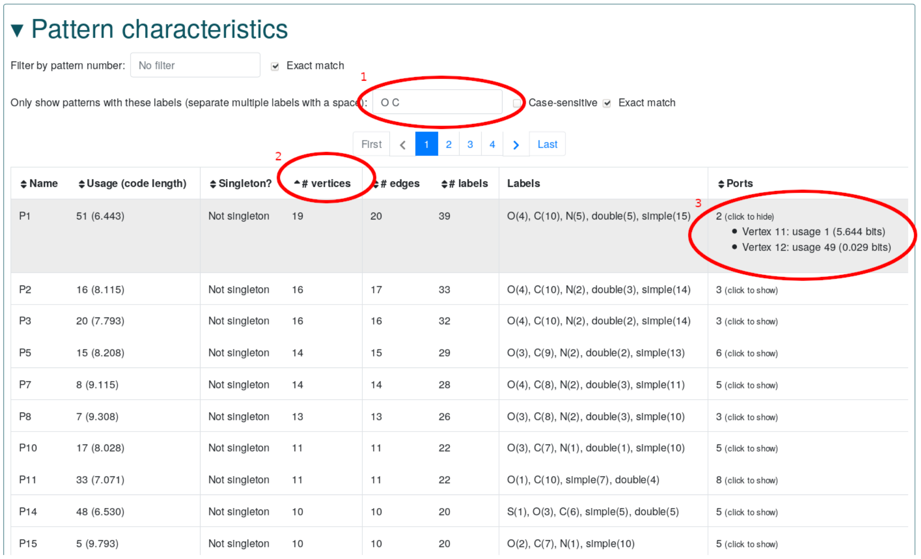
Pattern usage (no singletons)
Toggle box plot ●
25%: 7, Median: 17, 75%: 32.5
Min: 1, Max: 431
50
100
150
200
250
300
350
400
Pattern vertex count (no singletons)
Toggle box plot ○
0
2
4
6
8
10
12
14
16
18
20
22
24
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
The vertical dotted line represents the median
Pattern edge count (no singletons)
Toggle box plot ○
0
2
4
6
8
10
12
14
16
18
20
22
24
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
The vertical dotted line represents the median
Pattern label count (no singletons)
Toggle box plot ●
25%: 7, Median: 11, 75%: 16
Min: 3, Max: 39
5
10
15
20
25
30
35

Fig. 4. The "Main statistics" block. This blocks shows the global characteristics of the selected patterns, without focusing on the individual patterns. In this image the user switched the top-left and bottom-right plots to box plots. The red dotted lines in the plots represent their medians.

of 33 or less (top-left graph), and that half the patterns have 5 edges or less (bottom-left graph).

3.3 Step 2: Pattern Characteristics Table

In the "Main statistics" block the user has observed the general distributions of pattern characteristics. We now expect them to become interested in the individual patterns and wanting more specific information. The "Pattern characteristics" block, shown in Fig. 5 provides such information. It contains a table, each row of the table corresponding to one of the patterns selected by GraphMDL. For



▾ Pattern characteristics

Filter by pattern number: No filter ☑ Exact match

Only show patterns with these labels (separate multiple labels with a space): O C □ Case-sensitive ☑ Exact match

⬍ Name	⬍ Usage (code length)	⬍ Singleton?	▴ # vertices	⬍ # edges	⬍ # labels	Labels	⬍ Ports
P1	51 (6.443)	Not singleton	19	20	39	O(4), C(10), N(5), double(5), simple(15)	2 (click to hide) • Vertex 11: usage 1 (5.644 bits) • Vertex 12: usage 49 (0.029 bits)
P2	16 (8.115)	Not singleton	16	17	33	O(4), C(10), N(2), double(3), simple(14)	3 (click to show)
P3	20 (7.793)	Not singleton	16	16	32	O(4), C(10), N(2), double(2), simple(14)	3 (click to show)
P5	15 (8.208)	Not singleton	14	15	29	O(3), C(9), N(2), double(2), simple(13)	6 (click to show)
P7	8 (9.115)	Not singleton	14	14	28	O(4), C(8), N(2), double(3), simple(11)	5 (click to show)
P8	7 (9.308)	Not singleton	13	13	26	O(3), C(8), N(2), double(3), simple(10)	3 (click to show)
P10	17 (8.028)	Not singleton	11	11	22	O(3), C(7), N(1), double(1), simple(10)	5 (click to show)
P11	33 (7.071)	Not singleton	11	11	22	O(1), C(10), simple(7), double(4)	8 (click to show)
P14	48 (6.530)	Not singleton	10	10	20	S(1), O(3), C(6), simple(5), double(5)	5 (click to show)
P15	5 (9.793)	Not singleton	10	10	20	O(2), C(7), N(1), simple(10)	5 (click to show)

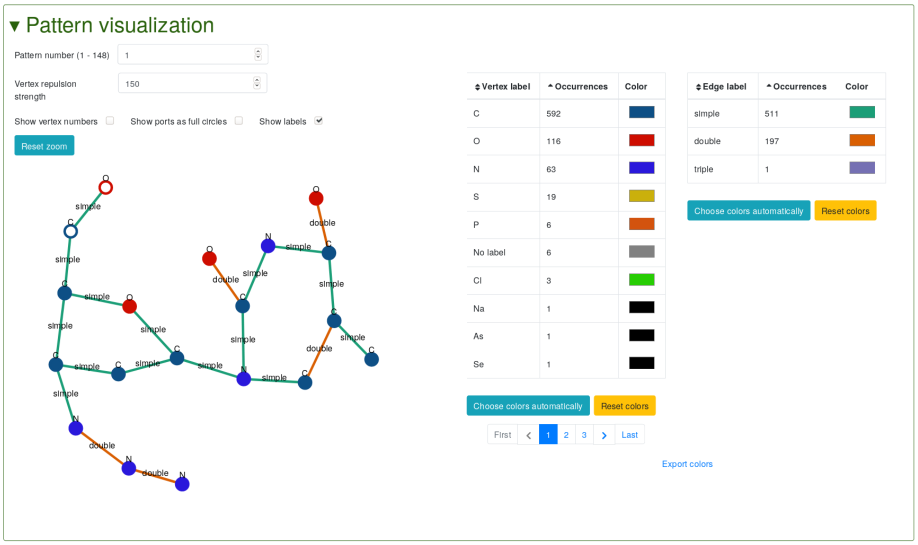
Fig. 5. The "Pattern characteristics" block. This block shows some information for each pattern. 1) A filter allows to only show the patterns that have certain labels. 2) Patterns can be sorted following their characteristics. 3) It is possible to show and hide detailed port information.

each pattern, it reports its usage, whether the pattern is a singleton[6] or not, its number of vertices, edges and labels, which labels are included in the patterns, and the number of ports it has. It is also possible to show in details which of its vertices are ports and how much each one is used.

The table is interactive. It is possible to filter the rows so that only patterns having some specific vertices or edges are shown, and it is also possible to sort the table by any of its columns, thus offering different points of view. For example in Fig. 5, the user chose to only show patterns containing at least an atom of Oxygen and an atom of Carbon. Then, the user sorted the table by number of vertices, so that the biggest patterns would be on top.

Thanks to this table, the user has now identified a pattern that interest them: pattern P1 (highlighted in Fig. 5). This pattern is used 51 times, has 19 vertices, 20 edges and a total of 39 labels. It has two ports: its vertices 11 and 12, which are used 1 and 49 times respectively. The user has now become interested in the structure of this pattern: a double-click on the pattern's row takes them to the next block.

[6] A singleton is either a single vertex with a single label or two vertices without labels connected by a single edge. They are the most basic patterns that GraphMDL can handle.



Pattern number (1 - 148): 1

Vertex repulsion strength: 150

Show vertex numbers ☐ Show ports as full circles ☐ Show labels ☑

⬍ Vertex label	▴ Occurrences	Color
C	592	🟦
O	116	🟥
N	63	🟦
S	19	🟨
P	6	🟧
No label	6	▨
Cl	3	🟩
Na	1	⬛
As	1	⬛
Se	1	⬛

⬍ Edge label	▴ Occurrences	Color
simple	511	🟩
double	197	🟧
triple	1	🟪

Fig. 6. The “Pattern visualization” block. The user can select a pattern, and the structure of the pattern will be shown. Colors of vertices and edges, as well as their positions, can be customised.

3.4 Step 3: Pattern Visualization

Graph are complex data structures and —especially for large graphs— visualizing them allows to better understand them. The “Pattern visualization” block, shown in Fig. 6, allows the user to display the structure of the patterns selected by GraphMDL, as interactive plots. The user can zoom on specific parts of the pattern if needed.

On the right, the user can choose the color associated to each vertex and edge label to customize the visualization. Color choices can be exported and imported, to make it easier to be reused. For example, the colors shown in Fig. 6 are based on the classic colors used when visualizing molecules. The user can also toggle the presence of vertex and edge labels on the plot. It is also possible to choose whether to highlight the port vertices of the pattern by showing them as rings instead of circles (e.g. the top-left vertices in Fig. 6).

Graphs are generally difficult to plot properly and in the way that the user expects. We implemented a drag behaviour in this block so that the user can click and drag vertices to move them. For example, if they prefer Oxygen atoms to be towards the top of the image, they can drag them to that position. This has the added benefit of allowing the users to “correct” the visualization when it turns out “messy”, like in Fig. 7. This interactivity really eases interpretation of the more complex patterns.

In Fig. 6 we can observe the structure of the pattern P1 that the user selected when analyzing the pattern characteristics table during the previous step. It is a fairly complex molecule with 19 vertices, which would be difficult to imagine

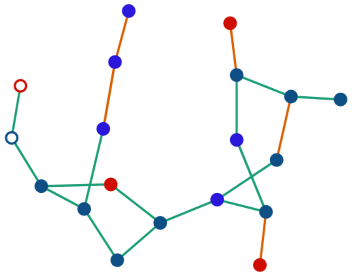


Fig. 7. Sometimes the visualization is not very clear, like above. Luckily, the user can click and drag vertices to help the application.

without a plot. Thanks to the visualization we can also easily identify that the pattern can be connected to other patterns only from one of its "branches": the one on the top-left, where its two ports are.

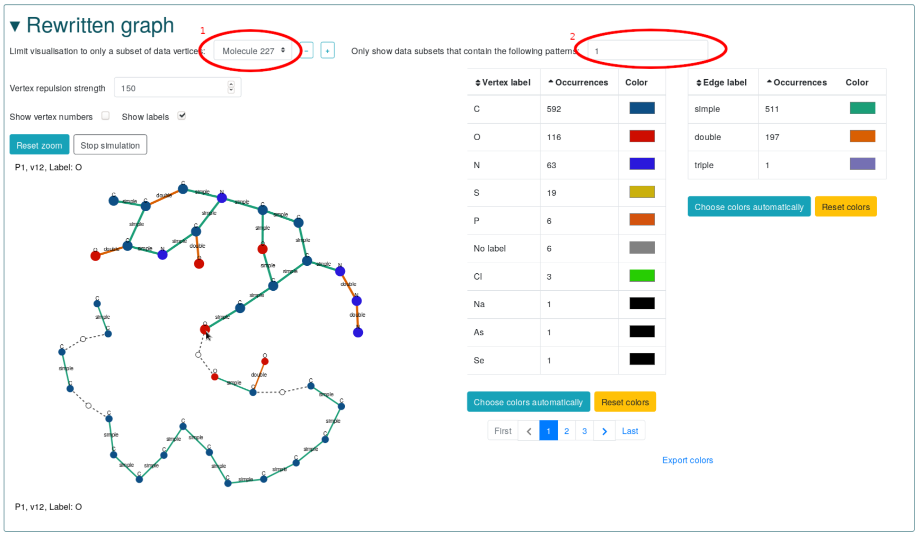
3.5 Step 4: Rewritten Graph Visualization

After having chosen and visualized a pattern, we expect the user to ask themselves how the pattern is used by GraphMDL to "cover" the data. The "Rewritten graph" block, shown in Fig. 8, visualizes GraphMDL's *rewritten graph* i.e. how the data is encoded as pattern occurrences. Since the data may be large, it would be very difficult to show the whole of the data at once: it would be difficult for the user to analyze and would probably require too much computation for the web browser. Therefore, we implemented the possibility of limiting the visualization to only some subsets of the data: the user can upload a "data subsets" file describing them. For example, the AIDS-CA dataset used as our example is a collection of molecules, each molecule being a separate connected component in the data graph. Thus, we created a data subsets file that allows to show only one molecule at a time. It is also possible to filter the list of data subsets via the interface, to only show the ones that contain specific patterns.

We found that the visualization ran smoothly on graphs up to some hundreds of vertices and edges. For larger graphs, we warn the user and ask them to confirm before visualizing the graph, in order to avoid an unexpected application freeze. In those cases, the user can instead decide to visualize a smaller part of the data by tweaking the data subsets file.

Fig. 8 shows the visualization of one of the molecules in the dataset, which contains pattern P1. This visualization is similar to the pattern visualization block. The users can zoom on the visualization, click and drag vertices to move them around, show/hide the labels and customize colors. In particular, the color customization of the two blocks are synced, so that the choices made when visualizing the data are maintained when visualizing individual patterns.

Thanks to this visualization, we can observe that the molecule shown in Fig. 8 is described by GraphMDL using 5 patterns, with pattern P1 describing



▾ Rewritten graph

Limit visualisation to only a subset of data vertices: Molecule 227

Only show data subsets that contain the following patterns: 1

Vertex repulsion strength 150

Show vertex numbers ☐ Show labels ☑

P1, v12, Label: O

Vertex label	Occurrences	Color
C	592	🟦
O	116	🟥
N	63	🟦
S	19	🟨
P	6	🟧
No label	6	⬜
Cl	3	🟩
Na	1	⬛
As	1	⬛
Se	1	⬛

Edge label	Occurrences	Color
simple	511	🟩
double	197	🟧
triple	1	🟪

P1, v12, Label: O

Fig. 8. The “Rewritten Graph Visualization” block. It visualize how the patterns are used by GRAPHMDL to describe the data, with an interface similar to the “Pattern visualization” block. 1) The user used the “data subsets” feature to only visualize a subset of the data. 2) Additionally, the user filtered the list of data subsets to only show the ones that contain pattern P1.

the main “bulk” of the molecule. By observing this visualization, the user will probably become interested in a new pattern (for example one of the other patterns that appear alongside P1). A double-click on a pattern takes the user back to the pattern visualization block showing the structure of that new pattern. The user can also just go back to the pattern table or the main statistics block, and change some options in order to continue exploring GRAPHMDL results!

4 Conclusion

In this demo paper we presented the GraphMDL Visualizer tool, which visualizes the results of the GRAPHMDL algorithm [4] in an interactive interface, to help users to better analyze and understand them. The tool presents itself as a web page, where the users can upload a result file generated by GRAPHMDL. The users can then navigate the four blocks that make up the visualization, each one responsible for a different step of the analysis process, and interact with them to customize the visualization to suit their needs. Throughout the paper we followed the analysis process of a user, highlighting the steps where our tool helped them understand the data, showing how an interactive visualization can help them.

While GraphMDL Visualizer has been designed for the purpose of visualizing GRAPHMDL results, it could be adapted to visualize results from other graph mining methods, as long as these results are converted into the format accepted by the visualizer. Of course some features of the visualization may not be available if the other mining solution does not support them (e.g. the notion of ports).

References

1. Build fast, responsive sites with Bootstrap, `https://getbootstrap.com/`
2. D3.js, data-driven documents, `https://d3js.org/`
3. Vue.JS, the progressive javascript framework, `https://vuejs.org/`
4. Bariatti, F., Cellier, P., Ferré, S.: GraphMDL: Graph Pattern Selection Based on Minimum Description Length. In: Advances in Intelligent Data Analysis. pp. 54–66. Springer (2020)
5. Grünwald, P.: Model selection based on minimum description length. Journal of Mathematical Psychology **44**(1), 133–152 (2000)
6. Jentner, W., Keim, D.A.: Visualization and visual analytic techniques for patterns. In: High-Utility Pattern Mining, pp. 303–337. Springer (2019)
7. Nijssen, S., Kok, J.N.: The Gaston tool for frequent subgraph mining. Electron. Notes Theor. Comput. Sci. **127**(1), 77–87 (2005)
8. Rissanen, J.: Modeling by shortest data description. Automatica **14**(5), 465–471 (1978)
9. Vreeken, J., van Leeuwen, M., Siebes, A.: Krimp: mining itemsets that compress. Data Min. Knowl. Discov. **23**(1), 169–214 (2011)
10. Yan, X., Han, J.: gSpan: Graph-based substructure pattern mining. In: IEEE Int. Conf. on Data Mining (ICDM). pp. 721–724. IEEE (2002)