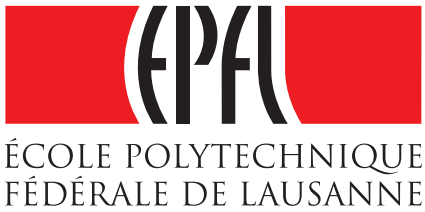


School of Computer
and Communication Sciences

Master Thesis

Intelligent Clustering for Graph Visualization

Lionel Martin
lionel.martin@epfl.ch

Academic Supervisor
Prof. Pearl Pu
pearl.pu@epfl.ch
Institute of Core Computing
Human Computer Interaction

Company Supervisor
Dr. Geraldine Bous
geraldine.bous@sap.com
SAP Labs France – Research
Business Intelligence Practice

February 20th – August 17th, 2012

Acknowledgments

I'm really grateful to Geraldine Bous, my supervisor at SAP, for all the support and the ideas we shared during this internship. It was for me a very good experience and the occasion to learn many things and meet very great people. I want to thank them all for the support and the friendly moments we had. I'm also grateful to Pearl Pu, my EPFL supervisor, for introducing me to User Experience and Human Computer Interactions.

Contents

Abstract

More and more areas use graphs for the representation of their data because it gives a connection-oriented perspective. Unfortunately, datasets are constantly growing in size, while devices have increasingly smaller screens (tablets, smartphones, etc). In order to reduce the quantity of elements displayed on screen, several techniques of information reduction can be used. Among them is graph clustering, which aggregates the elements of the original graph into clustered nodes and edges — thereby leading to a smaller graph.

In this report, we present a tool for the interactive exploration and analysis of large clustered graphs. The tool empowers users to control the granularity of graphs either by direct interaction (collapsing/expanding clusters) or via a slider that automatically computes a clustered graph of the desired size. In a next step, we explore the use of learning algorithms to capture graph exploration preferences based on a history of user interactions. The learned parameters are then used to modify the action of the slider in view of mimicking the natural interaction/exploration behavior of the user.

1 Introduction

Social networks, technology, biology, business data are part of the various fields that are studied with graphs. The intuitive interpretation of the data represented implied a need to develop techniques of graph analysis for these domains. This interest to graphs is pretty recent compared to the first time we heard about them. Indeed, the origins of graph theory dates back to the 18^{th} century. Since then a lot has been learned about graphs and their mathematical properties but it is only in the end of the 20^{th} century that they have become extremely useful as representation of systems in different areas.

The interest of users to represent information as graphs had created a need to research for manners to display those graphs. But these users had different constraints and then different needs. This is why the field of graph visualization is as large and the differences concern everything, even the most elemental part of the graph since the representation of a node and an edge vary. Indeed, even if a graph is most of the time represented as a set of points and lines, some researchers focused on displays based on the adjacency matrix [4] or even hybrid representations [10] (fig. 1).

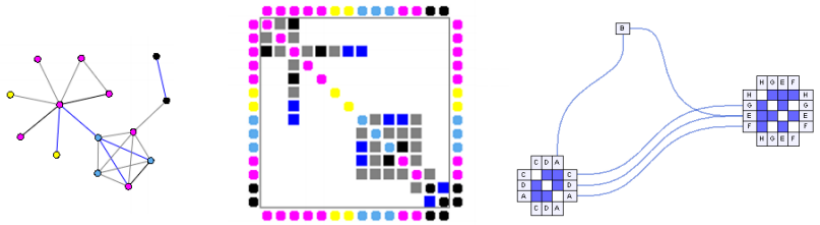


Figure 1: Visual representations of a graph. From left to right, a node-link, an adjacency matrix and a hybrid diagram [10].

Since graphs of all sizes and structures can be generated from datacubes or relational databases, we can obtain data which is not easy to interpret even if in general graph modeling is more intuitive than most of the other representations of data (fig. 2). For instance, if our graph has hundreds of edges, how can we visualize it efficiently? Indeed, with a limited space such as a screen we won't be able to display the whole graph if we want to gain insight or results from it, even with the best visualization technique. In this situation, choices have to be done to understand the information contained in the graph. These choices are left to the user which will be able to explore the graph interactively. These exploration tasks can be of different nature and grouped in two sets. They can be topology-based where the user-interest is related to the structure, such as, e. g., finding adjacent nodes, or determining connections between nodes. Alternatively, they can be attribute-based, in which case searching for nodes with specific values, and finding edges of certain types are frequent interests. For each task, one or more interaction techniques can be employed. Standard interaction techniques such as zooming, distortion or highlighting are commonly used in graph visualization. In addition, specialized techniques have been developed for interactive visual graph navigation and exploration [20].

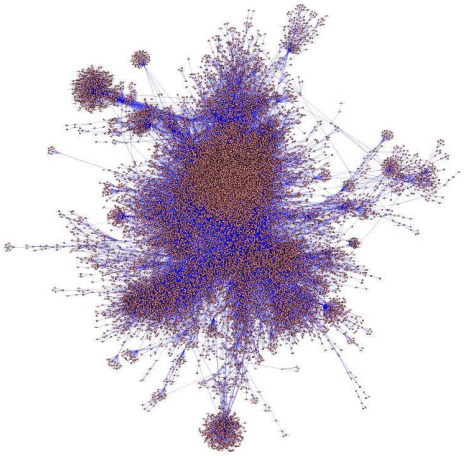


Figure 2: A graph with a high number of elements displayed.

Now consider graphs of millions of nodes. The techniques presented above cannot solve the problem alone anymore given the time it will take to the user to interact with such a huge graph. For this reason, we are forced to use also automatic techniques to change the structure of the graph. Given the quantity of data, we decide no longer to keep every single information displayed and we have multiple ways of doing so, two being the most famous ones. First one is clustering, where we aggregate the elements of the original graph that seem similar into entities which are their representatives in a new graph containing only those representatives elements. The second possibility is filtering, where we simply delete a part of the graph in which we have no interest.

The clustering techniques do not display original information anymore but keep every piece of information reachable by exploration. Thus a user can still interact with the graph, for instance by expanding the cluster that hides the data that it represents, to look deeper into one part of the structure, or by collapsing uninteresting data in its cluster (fig. 3). Nonetheless, we didn't find yet neither a good universal threshold for the quantity of elements to display to the user, nor which elements should be displayed. For these purposes, we try to improve user experience in graph visualization, using tools that mix different interaction methods on graphs and reproducing user behavior with the automatized techniques of information reduction.

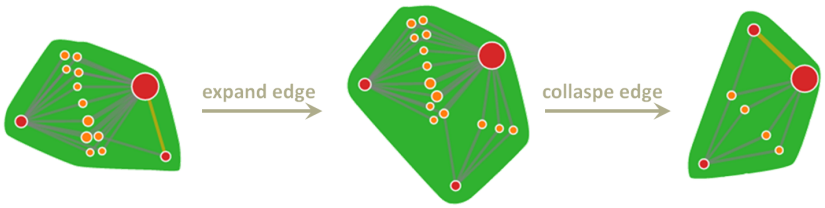


Figure 3: Expand and collapse interactions on a clustered graph.

The report is structured as follow. Section 2 presents the main techniques used for reduction of information. This section is the basis of all the possibilities we have for our research concerns. Section 3 introduces our use-case and the tools that have been implemented for the users in order to interact. In section 4 we detail one of the tools, combining the methods for information reduction presented before as well as user interactions. Section 5 explains the technique used to learn from the interactions of the user and presents tests for the exploitation of our model. Finally, section 6 concludes this work and proposes challenges for the future.

2 Information Reduction in Graphs

In this section we first present the most common clustering and filtering techniques for graph reduction. Afterwards, we focus on interactive data modeling.

2.1 Graph Clustering

For the last decades, clustering has been an intensively researched field proposing now a very large choice of methods for various kind of graphs. Since there is not a unique definition of what is a cluster, we are facing a large amount of work emphasizing various aspects of the graph in the simplification. Nonetheless, one can separate the clustering techniques into two main categories : structure-based and attribute-based algorithms. The former looks at the connectivity of nodes, structural properties like the existence of complete subgraphs, etc. to decide how to group the nodes. The latter is only concerned by the values of the attributes and the meaning of the data. However both have something in common, namely they want to aggregate nodes and edges in order to reduce the number of displayed elements. For a complete presentation of the different methods, the reader should refer to [6]. However, the most common ones as well as the ones that we are using are presented below.

Graph partitioning aims at minimizing the number of inter-cluster edges, which are edges connecting two nodes belonging to different clusters, given the intended number of clusters and their desired size. These methods, as well as many others in structural clustering, require the graph to be sparse (nodes must have low connectivity) to achieve good results. Otherwise, we would have a graph where any clustering would produce a lot of inter-cluster edges. Kernighan and Lin invented in 1970 a method which is still used nowadays but often in complement of another faster technique [13] (fig. 4). The algorithm solves bi-section problem only, as it is usually the case for this family of algorithms, and users that want more than two clusters have to apply the algorithm iteratively on the resulting clusters.
But this family of methods for clustering has another consequent drawback besides computational complexity: the fact that the user should give as input the number of clusters and even their size which is often impossible to predict, mostly if the graph has many nodes. Moreover, partitioning in more that 2 clusters is a bit approximative since iterative applications of the algorithm force the user to decide in advance the hierarchy of the clustering. For example, for

3 clusters the user needs to predict which one of the 2 first clusters should be partitioned again.

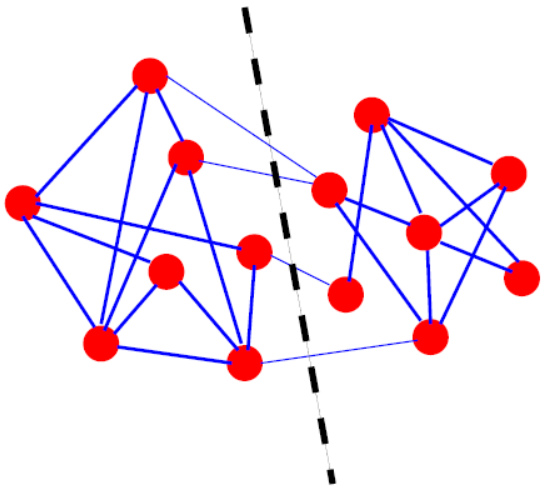


Figure 4: Graph partitioning on a sample graph. Only 4 inter-cluster edges remain.

As said previously, we don't have enough information to apply the graph partitioning techniques described above in general. For this reason and to follow common intuition about clustering, *Hierarchical clustering* methods were created. Their goal is simply to create a hierarchy with the clusters. Girvan & Newman invented one of the most famous techniques of this kind [7] (fig. 5) in which they proposed to start with the original graph and remove nodes one at a time to produce the clustering. To do this, they first select a similarity measure named *edge betweenness*, associating to each edge the number of shortest paths between any two nodes passing through it,

$$BC(e) = \sum_{u,v \in V} |\sigma_{u,v}(e)|.$$

Then, they remove the edge with highest score and recompute for all the remaining edges the measure and continue to delete edges. Intuitively, the highest score on this measure implies that the edge is a bottleneck in the graph, separating the vertices in several sets. Many other measures have been used such as counting the number of triangles an edge share [18] or simply based on the distance (stored as weight of edges) between the nodes for example. At the end of the process, we obtain a dendrogram representing the order in which the nodes were separated. To determine at which step we better have to stop the process, another measure was invented by Newman and Girvan a couple of years later: the modularity Q [17], defining the strength of a particular clustering with respect to a null model where the degree distribution is conserved but the edges are rewired at random. Probabilistic methods allow us to compute this measure once only, even if it will compare the graph with any random graph with the

same degree distribution, as follows:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{i,j} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j). \tag{1}$$

where m is the total number of edges $|E|$, A is the adjacency matrix of the graph, k_i the degree of node i, c_i its cluster and $\delta(c_i, c_j)$ ensure i and j have the same cluster. Thus, the higher this measure is, the better the clustering should be.

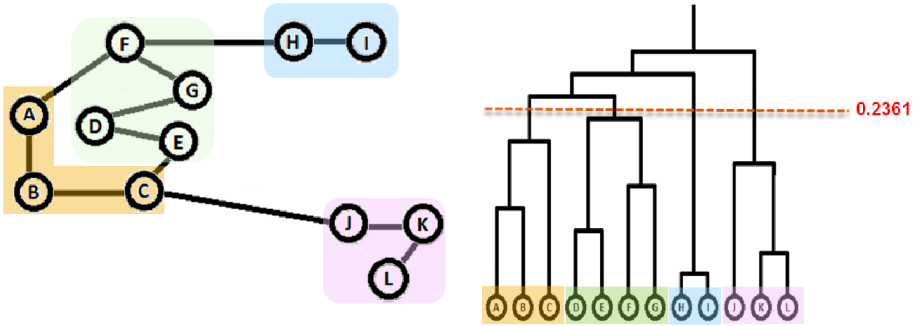


Figure 5: Application of the Girvan & Newman's method on a small graph.

Methods that remove edges starting with the original graph, like this one, are called *divisive* but the converse also exist: *aggregative* algorithms. One aggregative technique is, by the way, mainly based on the modularity score [16]. Indeed, one starts with the n disconnected vertices and for every pair of vertices, the change of the modularity score assuming we will connect them, ΔQ, is computed. At each step, the connection implying the highest change is performed and ΔQ values are computed again until all nodes are connected. At the end, the components can be considered connected in the dendrogram in two different manners. Either if they share an edge (*single-linkage*) or if they produce a clique, a subgraph where all the nodes are connected with each other (*complete-linkage*). Once again, the resulting dendrogram can be cut at the level which maximizes the modularity.

The algorithm by Newman is only one example but several other methods use this measure to determine the clustering from scratch. We call these techniques *Modularity-based clustering*. Without being exhaustive, we should at least present the fact that the modularity-based clustering is separated in three classes: greedy algorithms, such as Louvain's method [2], which are fast but not perfectly accurate, probabilistic methods like Simulated Annealing used in [8] really slow but accurate, and those employing extremal optimization heuristic like [3] which are generally good trade-off between time and result. We should also stress the fact that particular configurations of the graph may reveal false positive and false negative clusters using only this measure because it is only based on a particular null model and its statistics about the distribution. This issue is presented in more details in [6]. Depending on the graph and our interest, Louvain's technique might be very useful since it produces a hierarchy of

the clusters during the procedure that was not proved to be efficient but seems at least to be representative of the structure of the original graph. Furthermore, this algorithm is very fast, because even if a worst-case running time is not defined yet, an evaluation with 150 millions of nodes and a billion of edges took only 152 minutes to be clustered, reason why we will use it in our work. We should add that this modularity measure, as well as the methods using it, has many variations. Fortunato [6] presents the other possibilities as well as the limitations of modularity-based clustering in details. He mentions for example similar measure taking into account directed edges, weighted graphs, different null models, etc.

All the methods presented so far are here discussed for undirected, unweighted graphs. Nevertheless, extensions for directed or/and weighted graphs are straightforward. The often used idea for directed graph is to use in-degree and out-degree instead of the total degree and to replace the adjacency matrix by the weights matrix for weighted edges when needed.

There also exist methods based on the attribute values of the nodes in the graph. This is the case of *Dimension reduction* [21] where multivariate data is clustered regarding the categories of exactly two attributes (fig. 6). This technique allows to discover new connections because we concentrate on particular attributes but if nodes have many attributes, the user should test a big number of couples to understand completely the graph. Note that in this case, dense graphs can be considered and that sparse graph can be unreadable because the resulting clustered graph depends of the number of categories for the selected attributes. A really dense graph with few categories will be easier to analyze than a sparse one with too many possibilities for the attributes. Moreover, attributes which cannot be categorized such as results from a text analysis have no way to be taken into account.

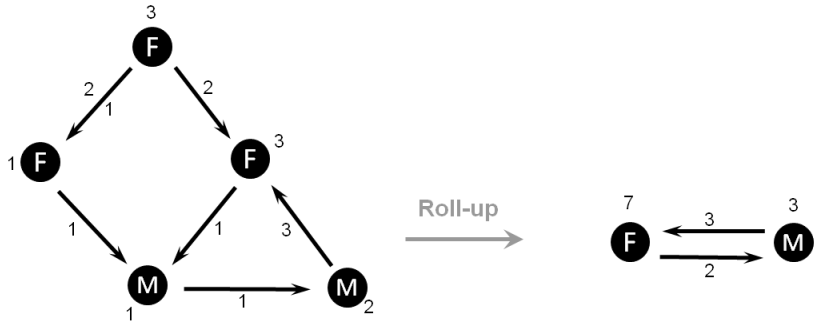


Figure 6: An example of application of the Dimension Reduction algorithm [21].

Interactive clustering is still a possibility, letting the user determine himself the clustering he wants. This approach allows to create hierarchies of clusters but has nothing automatized. The user selection can determine categorical differences or can match a regular expression [1], he can also separate the nodes given a structural criterion such as having high degree even if this usage is

really not common. Interactive clustering also contains expand and collapse operations we have already presented in the introduction (fig. 3).

2.2 Graph Filtering

Unlike clustering, filtering is often reduced to interactive user selection, either for structural or content interests. Moreover, it is often combined with clustering, in the sense that filtering is processed first and that clustering is then applied on the remaining data. Note that the converse occurs as well: clustering is performed and afterwards, only a subset of the clusters is conserved.

Nonetheless, few filtering methods exist to reduce the number of nodes of a graph based on structural properties. In this case structural measures are used to determine if a node is useful for the representation of the graph or the use that will be made of it, just like clustering. Huang [11] presented Node Importance Score in this perspective and Jia [12] proposed to use node Betweenness Centrality to determine the filtering (fig. 7). Both measures reduce the size of the graph by deleting the nodes that are not important enough, which means here that they are not connected enough or connected to nodes that are themselves not important. To avoid changing the meaning of the graph, the algorithms check however that the suppression of a node or its edges does not disconnect the whole graph. Note that connected components are treated separately from each other.

Another filtering concept is to produce a sampling of the graph that keeps the principal characteristic of the original graph such as the degree distribution. An evaluation [14] presents different way to produce this and shows that the most efficient ones are based on random walks and forest fires. But even if the results are good, this does not imply that this is what we are looking for. In general, the most interesting part of the data is the one containing the nodes that are the most connected to the rest of the network because those nodes might explain the relation. If we only keep half of the graph, then half of the most connected nodes will be deleted and so half of the nodes containing information too. But once again, it depends on the interest we have in the graph.

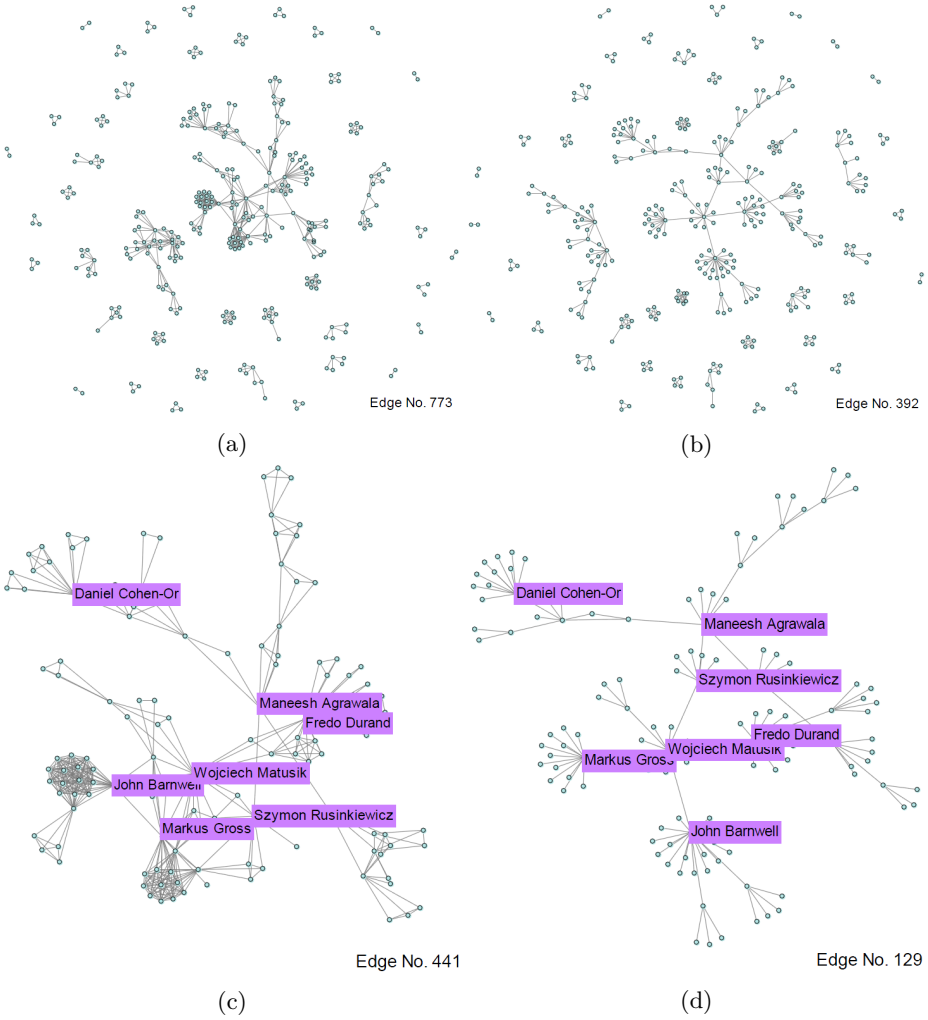


Figure 7: An example of filtering on a graph containing author collaboration. (b) and (d) are respective filtering of (a) and (c), the whole set and its largest component [12].

2.3 Interactive Data Modeling

The methods presented above for the reduction of information are standard popular techniques of clustering and filtering which are always based only on structural properties and attributes of the original graph. In the following we want to take advantage of the other properties one can derive from graphs to reduce the number of displayed elements.

Rather than displaying everything, we thought about a solution where the user decides what dimensions he is interested in. This solution presents several advantages, like the reduction of the number of elements displayed, improving the readability, but also take into account that different interest on the graph are sometimes better understood with different representations of the data.

For example consider the set of publications in scientific journals in 2010. If we draw a graph linking authors and papers on this period, we will easily count the number of publications of each author with the degree of its node. But counting the number of publications authors A and B did together is more complex because we need to search in all the publications of A whether B is a neighbor. However, the graph is bipartite since the relation between the nodes means "Author A is one of the authors of publication P" so neither two authors, nor two publications can be linked. So we can compute the paths of size two starting at an author in the previous graph and we will obtain a new graph where nodes represent only authors and the links represent the publications they have in common. This operation is called *projection* and can be done with any bipartite graph. Finally, with this new graph we can easily answer the second question we had before by simply looking at the weight of the edges (fig. 8).

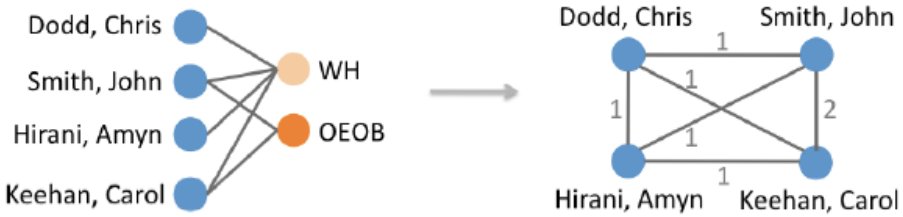


Figure 8: Projection [15].

Different tools are available for interactive data modeling (see [9, 15] and fig. 9) and given our interest in user experience for graphs, we reuse part of the idea. First, when facing a bipartite graph, user will be able to project one set on the other and second, when facing hierarchies, user will have the choice to select one layer in the hierarchy to be displayed rather than everything.

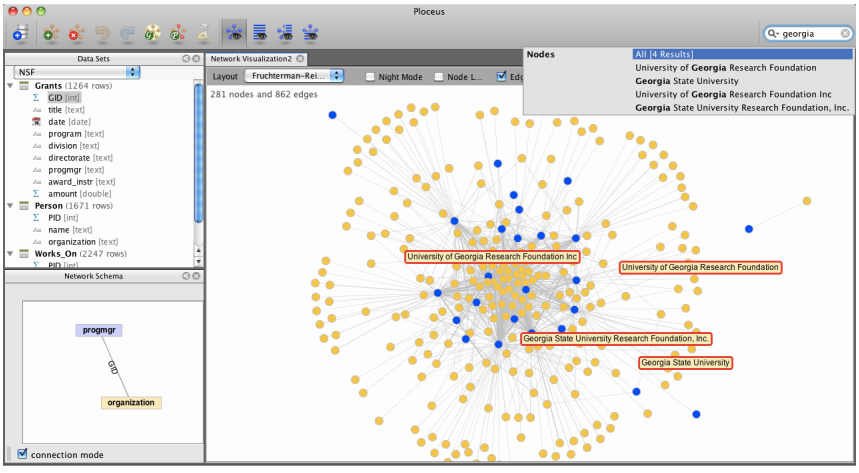


Figure 9: User interface of Ploceus with 3 different views: a management view on the top left, a network schema view on the bottom left, and a network visualization view on the right [15].

Let assume we can split the dataset into two groups, both being a hierarchy separated by a relation. An example is described in the figure 10 and represents the stores where employees worked last year. H_A and H_B are the two hierarchies; L_A and L_B are the sets containing the leaves of (resp.) H_A and H_B; finally R is the relation linking the two hierarchies. R is defined by 4 smaller relations : $R_{AA} \subseteq L_A \times L_A$, $R_{AB} \subseteq L_A \times L_B$, $R_{BA} \subseteq L_B \times L_A$, and $R_{BB} \subseteq L_B \times L_B$ to specify directed links between L_A and L_B and other links between the leaves of the same set.

Practically, those operations are made with adjacency matrices. We start by querying once the relational database containing the data and store everything in 3 different adjacency matrices : M_A, M_B, and R. They respectively contain the hierarchy structure in H_A, the one in H_B and the 4 relations together for R. All the matrices are square, the last one having $|L_A| + |L_B|$ rows and columns. Recall that adjacency matrix of a graph is a matrix whose values are 1 on line i and column j if node number i is connected to node number j and 0 otherwise. Note that if edges are directed, then elements (i, j) and (j, i) might be different. For example, L_A-projection, which result in the set of nodes of L_A connected if the two nodes are both connected to the same node of L_B, is computed as $\delta(R)^2$ restricted to the first $|L_A|$ rows and columns. The $\delta(M)$ operator takes the binary representation of the matrix M, replacing any positive value by 1, multiplying k times the same matrix gives the paths of length k and restricting the result to the first rows and columns reduces the part of interest to the links from L_A to L_A only.

Let's use again the example about the publications. Imagine you have a hierarchy containing the Authors and the Entities they are working for and an other one with the Publications and the Topics they are related to (only one topic each). The relation corresponds to "Author wrote Publication". With this, the user can ask: "What entity is concerned about what Topic ?". And thus we just have to compute the result of $M_A \times R_{AB} \times M_B^T$, M_A being restricted to the rows containing Entities and columns containing Authors and M_B to the rows containing Topics and columns containing Publications for size matching (fig. 11).

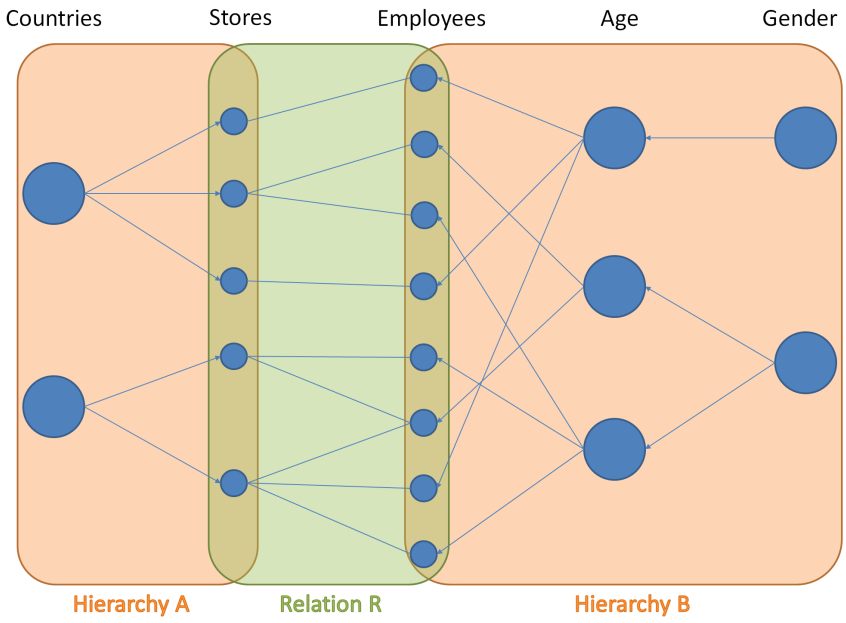


Figure 10: Representation of the structure of the hierarchies and the links between entities.

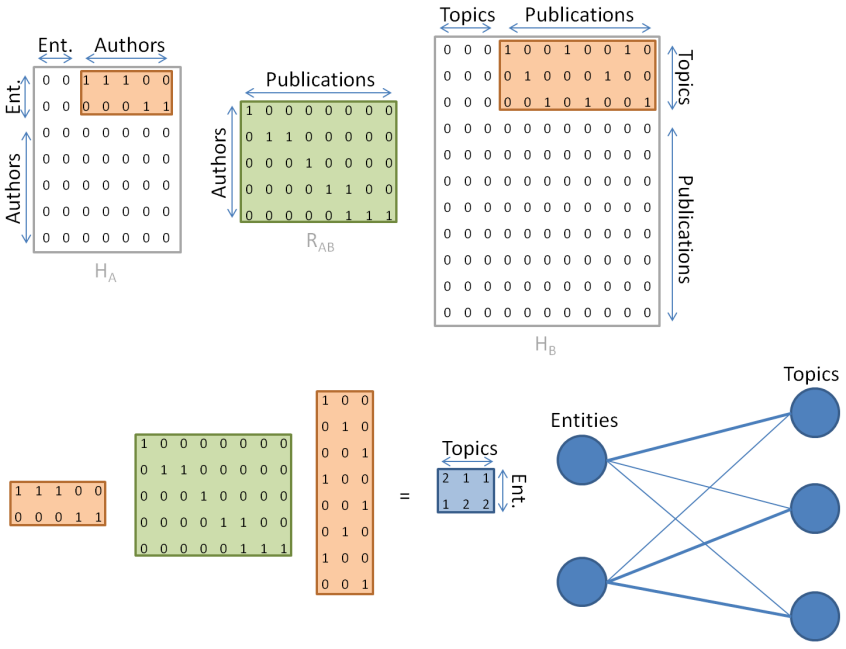


Figure 11: Matrix computations for the example presented above. The original graph is the same than part of the previous figure (without the Gender layer) that have been relabeled from left to right Entities, Authors, Publications and Topics.

3 Implementation

To illustrate all the forthcoming work about user experience, we present an example that we will follow all along. This represents data taken from an online forum.
We have a relational database containing Users, Messages, Threads and Forums. Each entry of the database represents a Message posted by an User in a particular Thread which is in a Forum. We have a hierarchy between the Messages, Threads and Forums (H_A) since every message is in exactly one Thread which is in exactly one Forum. The relation links a User to the Messages he has posted. We also have a reply structure that stores which Message is a reply to which other Message and then produces the relation R_{AA}. The database contains 32942 Users and 427221 Messages contained in 96337 Threads and 33 Forums but some tests are made over a subset of the database containing only 2247 Users and 351 Messages in 105 Threads and 7 Forums.

User Interface is presented in fig. 12. We have different tools to help the user explore the graph and interact with it:

(1) Menu Selection
(2) Filtering Tool
(3) Dataset Modeler
(4) Clustering Methods
(5) Information Slider

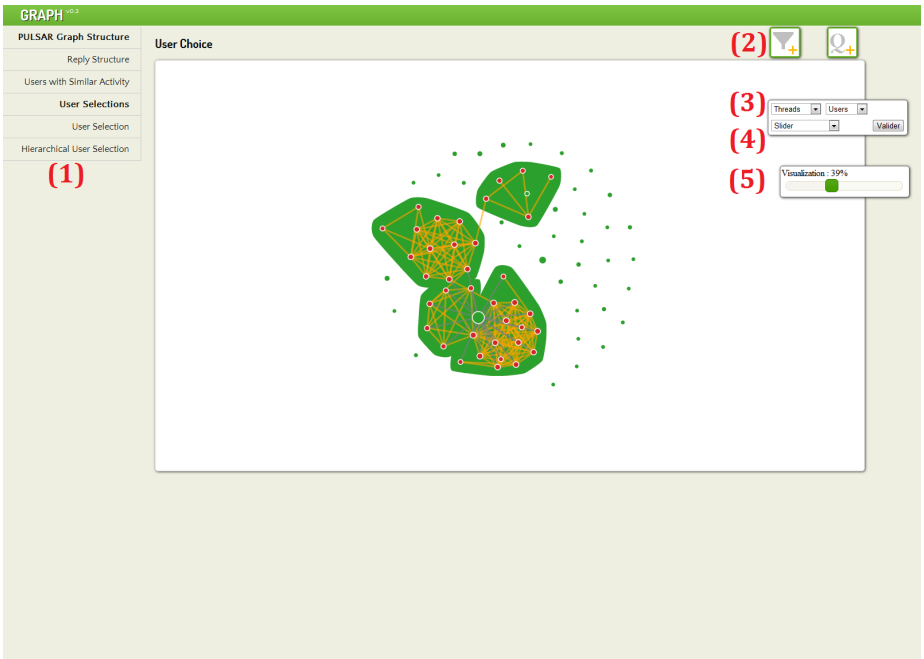


Figure 12: User Interface of our visualization tool.

3.1 Menu Selection

The starting web page of the application contains 4 different menus. The two first ones are linked to a query and its parameters that is sent to the database and whose result is parsed in order to obtain a graph. This is the simplest execution mode where everything is prepared and no interaction but filtering is possible after selection.
Then, the two last ones are different attempts to do interactive modeling of datasets. The third one lets the user choose between a list of already defined possibilities that can be queried. This is the solution half way between the completely static system described above and the completely interactive system we could imagine. Finally, the last one works differently. The two hierarchies of the use-case are queried once and stored on the server and then the user can choose to display the layer he wants with drop-down lists. Any combination of two sets is allowed as long as it is not a trivial query. For example, we refuse to display connection between two layers of the same hierarchy since it will just show the links of the hierarchy. On the other hand, we allow connection between a set S and itself, which we interpret as a S-projection over the leaves of the other hierarchy. For example, if user wants to see "Forums and Forums", then we display the Forums-projection over Users: the Forums are linked if there exist at least one User that posted something in both Forums.

3.2 Filtering Tool

Although this work focuses on clustering for its automatized aspects, we can filter data using the *Filtering tool*. With this tool, we can omit particular nodes, select subsets of the data given particular specifications for parameters or reduce the number of measures that we display as edge weights.

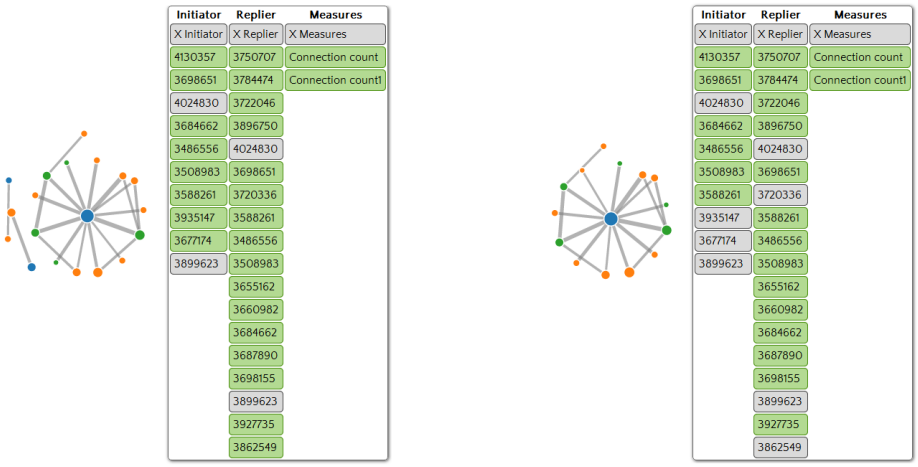


Figure 13: User filtered the two links that were not part of the largest component.

3.3 Dataset Modeler

As explained earlier, when the user chooses the dataset modeling option in the menu, the data is stored in adjacency matrices representing the hierarchies and the relation between the leaves of the two hierarchies.
Given the tree structure of the hierarchies and the few connections between the leaves of the two sets, adjacency matrices are sparse matrices, stored as a list of values with respective indexes. Then, when the user selects two sets to display, operations on the adjacency matrices give the path of correct length between the desired sets and display information in a bipartite graph. These operations are simply matrix multiplications between the different adjacency matrices that were described at the end of section 2.3.

3.4 Clustering Methods

With this, the user can choose between several different information reduction techniques. The possibilities are *projection*, *random filtering*, *Louvain's method* and the *information slider* for now. All but random filtering require a layered representation of the data and to do this we needed a new structure for the nodes and the edges, called respectively GraphNode and GraphEdge (fig. 14).

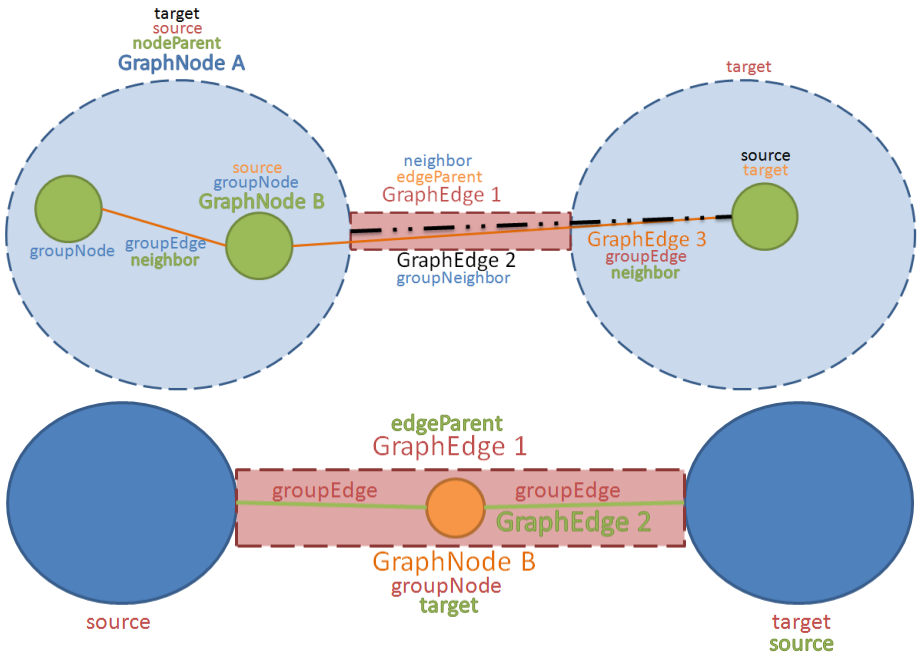


Figure 14: Details of the structure of the GraphNodes and GraphEdges. The variable names are applied to the element with the same color. For example, GraphNode A is the source of GraphEdge 1.

Both GraphNodes and GraphEdges work on the same model. They are composed of identification variables (usually a name and an ID), a size, a group, optional parameters (e. g., the depth in the graph) and lists of other GraphNodes and GraphEdges to represent the hierarchy. Nodes and edges clusters

are represented with doubly linked lists: every node and every edge contains a parent variable and a children variable. More precisely, there are two parents variables: a nodeParent and an edgeParent (potentially a node can belong to several edge clusters — recall fig. 8) and two children variables: one for the nodes (groupNodes) and one for the edges (groupEdges) clustered in this element. Moreover, an edge contains its source and target and a node contains a list for its neighbors, which are simply the edges connected to it, and a list for induced neighbors produced by the clustering. The latter one is necessary to display links between nodes of different layers of clustering since neighbors are only links between nodes of the original graph (on the last layer) or between the two parents of the nodes connected by a link (on any layer but always the same for the two nodes). The reason we want to display nodes of different layers is because the user can expand particular nodes and then display data that is not on the same layer and we need a way to connect this data (fig. 15).

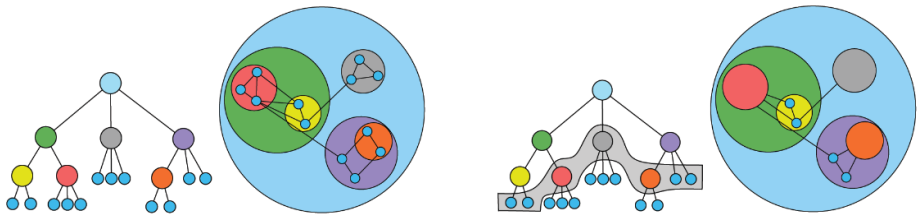


Figure 15: Representation of a user exploration of a graph and its dendrogram shows the importance of groupNeighbors [1].

With these structures, we can now describe Louvain's method and its implementation. Louvain's method, presented and tested by Blondel [2], is a fast efficient and often used modularity-based aggregative hierarchical clustering technique. This means that this technique is used to cluster the nodes of the graph in order to reduce the size of the final displayed graph. It is aggregative because at the beginning, all the nodes are considered separately as clusters of one node (themselves) and then, they are aggregated together. Finally, the measure used to determine how to cluster the nodes together and when to stop the process is the Modularity measure Q presented in more details in section 2.1 (eq. 1).
The Louvain method can be summarized as follows:

ALGORITHM:

1 - Repeat sequentially until no local improvement is possible

a. Assign a different community to each node of the network;

b. For each node i, consider the neighbors j and evaluate the gain in modularity that would be produced by placing node i in the community of node j;

c. Place the node i in the community for which the gain is maximum (if no positive gain, then don't move node i).

2 - Build a new graph

a. Let the nodes of this new graph be the non-empty communities resulting from phase 1;

b. Set the weights of the edges as the sum of the weights of the links between nodes in the corresponding two communities (creating self-loops if there were links between nodes that now belong to the same community).

3 - If the modularity score changed in the last pass, then return to phase 1

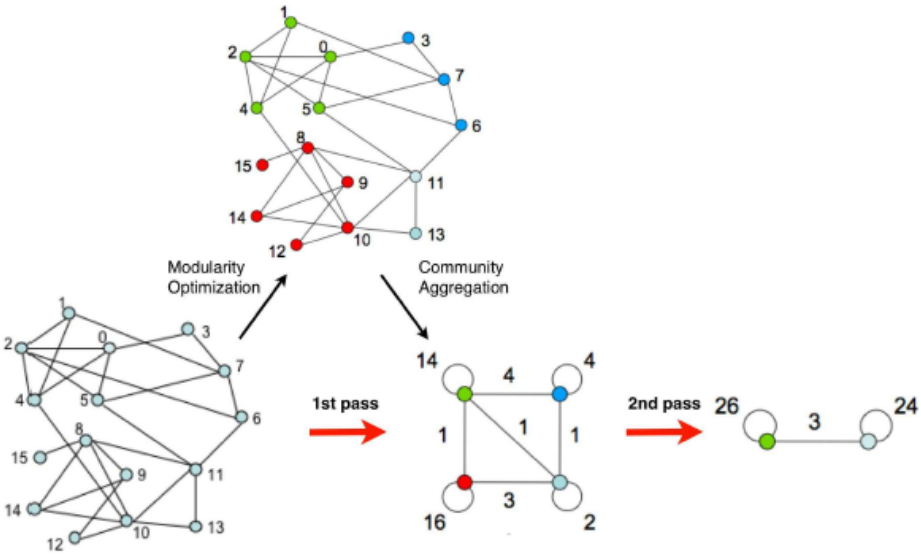


Figure 16: Two passes of the Louvain's method on a graph with 16 vertices and 28 edges [2].

NOTES:

- Phase 1 often implies that a particular node is selected several times during the same pass. Once the last node is considered, then the first one is considered again and the phase stops only if any improvement has been made for the last N nodes.
- Rather that computing the modularity value for the whole graph every time, we can simply compute the gain ΔQ for a node moving from community c to d. Blondel presented it in his publication [2] for an isolated node i that moves to community C:

$$\Delta Q = \left[\frac{\Sigma_{\text{in}} + k_{i,\text{in}}}{2m} - \left(\frac{\Sigma_{\text{tot}} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{\text{in}}}{2m} - \left(\frac{\Sigma_{\text{tot}}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right],$$

  where Σ_{in} is the sum of the weights of the links inside C, Σ_{tot} is the sum of the weights of the links incident to nodes in C, k_i is the sum of the weights of the links incident to node i, $k_{i,\text{in}}$ is the sum of the weights of the links from i to nodes in C and m is the sum of the weights of all links in the network.
- The order in which the nodes are considered in phase 1 can influence the computation time but seems not to be significantly influencing the modularity score obtained.

IMPLEMENTATION:

- In phase 1, the nodes are considered community by community and if a change occurred, then the remaining communities are not considered and the phase 1 starts again.
- Rather than computing the change of modularity sequentially for all the neighbor communities and keeping the largest value at the end, we prefer to compute the changes for all the communities, at the same time, neighbor by neighbor but not necessarily belonging to the same community. This reduces the number of computations and avoids search operations to retrieve the community of the neighbors.
- Starting from Q as defined in the literature, we will derive our gain ΔQ:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{i,j} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j),$$

where $A_{i,j}$ represents the weights of the edge between i and j, $k_i = \sum_j A_{i,j}$ is the sum of the weights of the edges attached to vertex i, c_i is the community to which vertex i is assigned, the δ-function $\delta(u, v)$ is 1 if $u = v$ and 0 otherwise and $m = \frac{1}{2} \sum_{i,j} A_{i,j}$.
So if a node i moves from community c to d, then Q increases for the nodes j in d and decreases for the nodes j in c:

$$\begin{aligned} \Delta Q &= \frac{1}{2m} \left[\sum_{j \in d} A_{ij} - \frac{k_i k_j}{2m} - \sum_{j \in c \setminus i} A_{ij} - \frac{k_i k_j}{2m} \right] \\ &= \frac{1}{2m} \left[\sum_{j \in d} A_{ij} - \sum_{j \in c \setminus i} A_{ij} \right] - \frac{k_i}{4m^2} \left[\sum_{j \in d} k_j - \sum_{j \in c \setminus i} k_j \right] \end{aligned}$$

- For the different passes, the data structure is made of a list of communities and a list of nodes from the original graph. The communities are here to aggregate the nodes together given the modularity and at the end of the phase, a new layer of communities and nodes is created. The structure is maintained as follows:

  1 - Create the set of nodes N_0, a copy of the original graph with no children and no parents;

  2 - Create the set of isolated communities C_0: GraphNodes with N_0 as children (and them as N_0's parents) and set $i = 0$;

  3 - Run phase 1 of Louvain's method and increment i;

  4 - Create C_i, for non-empty C_{i-1} with C_{i-1} as children of C_i;

  5 - Create N_i with parents C_i and children N_{i-1} (and N_{i-1}'s parents to N_i) and set C_i's children to N_i;

  6 - If ΔQ increased in the last pass, then go back to step 3.

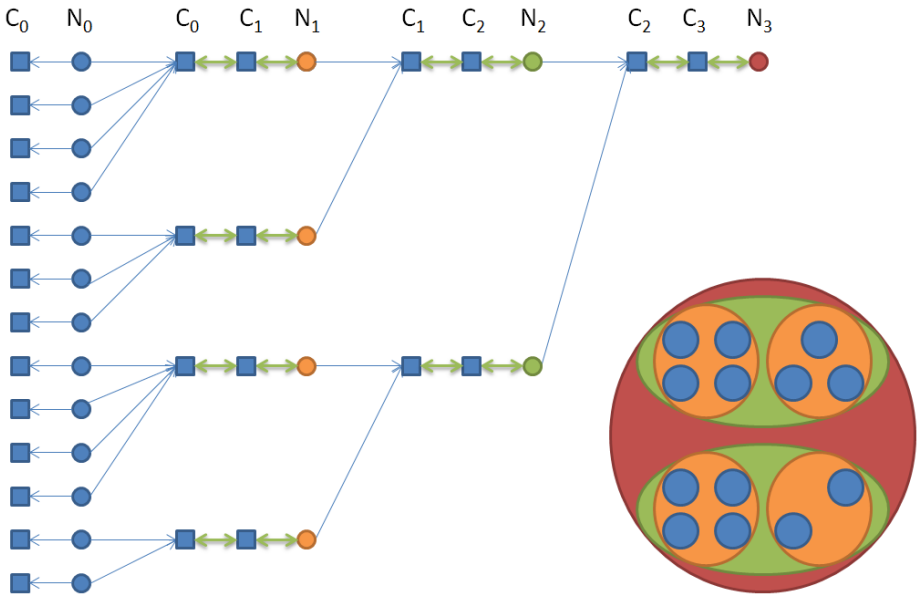


Figure 17: The steps of creation of the structure for a small graph and its clustered representation. Edges are omitted for simplicity.

3.5 Information Slider

The slider is another way to transform a user query into a graph. This time, the user will select the datasets he wants to see and ask for a proportion of information displayed as presented by Elmqvist [5].
To do this, he will use a slider going from 0% (minimal graph) to 100% (maximal graph) that will show the requested percentage of information.
By controlling this limit, we can either use it to ensure a minimum frame rate by capping the amount of visual entities to draw, or to control the amount of elements for the purpose of efficiently perceiving the visualized data.
In the following section, we present the system that recommends the elements of the graph that should be displayed based on the quantity of information that the user will request with this slider.

4 Recommendation system

Depending on the knowledge we have about the graph, we can manage to cluster and filter data in many ways. Structural solutions are preferred when we don't have much information and obviously we will try to take advantage of any particular information such as metadata. For example, if we have bipartite graphs we will try to do projections and if we know the attributes, we will cluster nodes with respect to those attributes or we will do dimension reduction. Nonetheless in all these situations, we still need to know user interests for better displaying...

Indeed, even if automated methods do a great job, the user might have interests that differ from their representation. For example, structural clustering

methods will often produce graphs that are not what he wanted to see because they are based on structural statistics and the users almost always care about the attributes. Then he will need to interact with the graph to explore it and obtain the best display with respect to his interests.
Here, we first want to give the user all the tools he could need to explore the graph and then try to recommend the best display. For this second part, we will have to understand the interactions of the user and find the best heuristic.

Let's consider a graph G containing a total of $|E|$ edges and $|V|$ vertices. We denote by $|E_i|$ and $|V_i|$ the number of displayed edges and nodes respectively at time step i. Our goal here is to provide to the user a slider that will allow him to determine what quantity of information he wants to see, using a precise *measure* (recall (5) of fig. 12). The measure is used to interpret the meaning of the amount of information displayed. It is a linear interpolation of the two extrema always given as a percentage: the maximum (100%) is defined by the total number of nodes and edges in the original graph and the minimum (0%) is defined by the smallest number of elements that we need to display after reduction to keep every information reachable by exploration. Different measures could be considered depending on the weights we give to the different nodes and the different edges, for example we could consider only nodes, only edges or any combination of the two. We assume that for now the information measure is

$$I_i = \frac{|E_i| + |V_i|}{|E| + |V|}, \tag{2}$$

giving as much importance to nodes and edges.

Given this definition, the measure is always between the most compact graph that can be obtained from the original graph with information reduction techniques and the original graph itself. Thus, we will need to reduce the more we can the size of the graph, using the different methods presented in section 2, while permitting the exploration of every single datapoint. Note that this prevents the use of filtering techniques which don't force every datapoints to stay reachable.

For our forum use-case (presented in section 3), we are sure to face either a bipartite graph or a projected graph so the first thing we will do is to project one set on the other if it wasn't asked directly by the user (by choosing the same set to display twice). This will reduce the number of nodes but might increase the number of edges, thus we should be careful while applying projection. Indeed, if a node of set B is connected to several nodes of set A, then A-projection will replace this subgraph by a clique of the nodes connected to the previous node of set B (fig. 18). Then, we continue the reduction with *Louvain's method* for clustering on the projected graph. This clustering algorithm has the advantages to be fast and applicable with almost no knowledge of the graph, since we just need to compute the Modularity score.

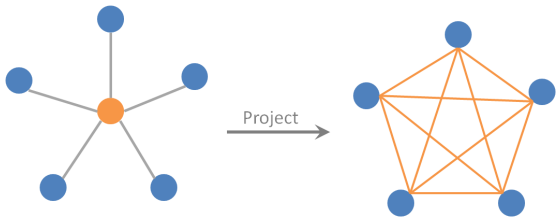


Figure 18: 6 nodes and 5 edges are projected into 5 nodes and 10 edges.

Using the information measure that we had define earlier in equation 2, the projection of a node with many neighbors won't reduce the amount of information displayed. Indeed, it will even increase it because as it was explained just before, the subgraph that was containing $N+1$ nodes and N edges representing a node and its N neighbors, will be replaced by a N-clique, containing N nodes and $\frac{N(N-1)}{2}$ edges. Obviously, letting I_i be the measure before projecting and I_{i+1} the measure after the projection, we have :

$$I_i = \frac{2N+1}{|E|+|V|}, \quad I_{i+1} = \frac{2N+N(N-1)}{2(|E|+|V|)}$$

$$\text{and then: } I_i > I_{i+1} \Rightarrow 2N+1 > \frac{2N+N(N-1)}{2}$$

$$\Leftrightarrow N+1 > \frac{N(N-1)}{2}$$

$$\text{whose solution is: } N < \frac{3+\sqrt{17}}{2} \approx 3.56$$

So in the cases where the largest projection contains more than 3 nodes and we want to reduce the information displayed by projecting (this is always our case), then we will need to introduce parameters α and β to change I_i into: $I_i = \frac{\alpha|E_i|+\beta|V_i|}{\alpha|E|+\beta|V|}$, where α and β need to satisfy:

$$\frac{\alpha N+\beta(N+1)}{\alpha|E|+\beta|V|} > \frac{\alpha\frac{N(N-1)}{2}+\beta N}{\alpha|E|+\beta|V|}$$

$$\Leftrightarrow \quad \alpha N+\beta(N+1) > \alpha\frac{N(N-1)}{2}+\beta N$$

$$\Leftrightarrow \quad \alpha N+\beta > \alpha\frac{N(N-1)}{2}$$

$$\Leftrightarrow \quad \beta > \alpha\frac{N(N-1)-2N}{2}$$

$$\Leftrightarrow \quad \frac{\beta}{\alpha} > \frac{N(N-3)}{2} \quad \text{and } \alpha > 0$$

For simplicity, we set $\alpha = 1$ and $\beta = \frac{N_{\max}(N_{\max}-1)}{2}$ in our use-case. It represents the number of links in the largest clique of the graph and it is easy to compute.

Once the bounds are set and the measure is defined, we will have to focus on the intermediate steps. Recall that the slider will simply help the user to choose the proportion of information to display given the measure associated with it. Then we need to define how to interpret this measure which only gives

us a number of elements to display without saying neither exactly which ones should be displayed, nor what proportion should be nodes and what proportion should be edges.

We can combine the interactive cluster exploration presented in the beginning of this report (recall fig. 3) with the modifications produced by the slider. Indeed, changing the proportion of information is the same as applying several (or sometimes just one) expand operations on the minimal graph, the starting point. It is the case because every operation made on the original graph in order to transform it into the minimal graph reduces the information measure, even the projections which increase the number of elements by definition of α and β. It can as well be a succession of expand or collapse operations on any graph resulting from exploration since those transformations can be concatenated and that for every interaction there exists an inverse that cancels it (collapsing a cluster cancels its expansion and vice-versa). Thus, we define *sequences* as the different lists we can create with those expand and collapse operations to transform an initial graph into another. If no reference graph is given, we assume that the starting graph is the most compact one, which will often be the case. We call also *configuration* the state of the graph after applying all the operations of the sequence, or until we decide to stop.

Furthermore, the principle defined above with the sequence of expand operations forces us to display all the nodes and edges contained in a cluster at once. If we are supposed to open a big cluster to improve the displayed information and that cluster aggregates a consequent part of the information ΔI, we have two possibilities for now: either we keep the cluster or we display all the aggregated elements. However, if the user moves the slider from I_i to $I_{i+1} = I_i + \delta \cdot \Delta I$, with $0 << \delta << 1$, then there is a gap between the query of the user and the two solutions. We want to find a way to fill this gap and we have several ways to do so. The first possibility is simply to decide to show only part of the contained information with respect to the value δ. A practical solution could be to set the cluster transparent but still visible and display part of the nodes and edges on top of it. Another possibility could be to redefine the clustering inside the cluster to expand in order to get the wanted quantity of elements by creating a new layer in the hierarchical clustering.
Note that sometimes, even with all those precautions, we can't avoid the gaps, either because there are a few number of elements to display in total or simply because the slider gives a continuous value to the information measure and that we have to translate this in a discrete measure, the number of elements to show.

One more comment is required about the measure of information we tried to define above. In practice, it will often happen that the measure only define a number of elements to display without defining a bijection between the amount of information and the precise elements to show, meaning that there could exist many configurations that represent the same amount of information.
Moreover, if there are several ways to reach a given quantity of information, there are also potentially different intermediate steps leading to the same configuration. For instance doing expand on node A and then on node B is a different sequence than B first and then A but the result is the same if A and B are disjoint sets of nodes and edges (see fig. 19). With all this, we could be able

to construct a huge number of different sequences that will lead to the same quantity of information so we will have to choose one as the *default sequence*. This sequence will be the one followed by the information measure and as long as the proportion won't be reached, more operations of this sequence will be executed. So, we now have to answer in what sequence we should expand the elements. Moreover, we need to find which elements should be displayed instead of the others for a given amount of information taking into account all the remarks above. Lots of parameters can be considered to determine the order in which the different elements are displayed using the slider. For example, based on the size of the nodes, the strength of the links, the number of neighbors, and so on.

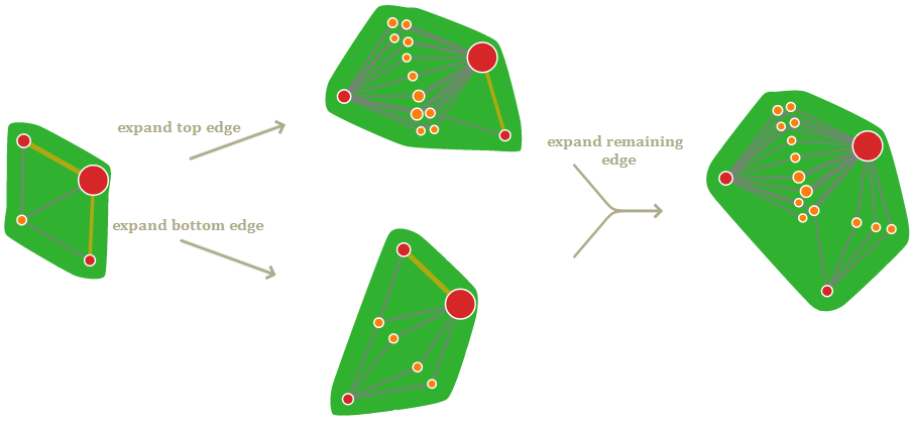
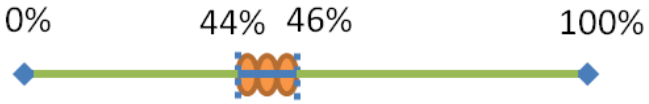


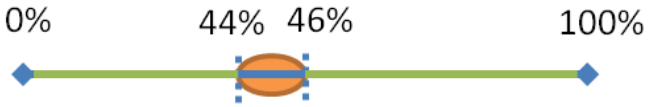
Figure 19: Two different sequences leading to the same configuration.

Finally, we want the user to be able to interact directly with the expand and collapse techniques too: when a cluster is displayed, the user can simply click on it to expand it (suppressing the cluster and showing all the nodes and edges contained inside). He also has a way to collapse what he has expanded before (fig. 3). Of course, the reason is that we want to allow the user to correct a clustering that wouldn't correspond to his interests in the graph, so we need him to be able to explore. But by doing so, he might change the quantity of elements displayed and thus the proportion of information represented by the slider. So, it is important to remark that the quantity can be affected by interactive operations on the graph.
The fact that the user can both use the slider and do interactive exploration forces us to solve a new issue. If the user starts using interactive exploration at some point and expands or collapses elements in a different order than expected by the default sequence (following a sequence different than the default one), then we won't be able to use the default sequence anymore for the incoming modifications that the user will request by the use of the slider. Indeed, there could be a gap between the configuration before the interactive clustering and the configuration after the exploration.
For example, assume that he uses the slider to display 44% of the information and then expands the 3 smallest node clusters displayed, C_1, C_2, and C_3. The new amount of information displayed is 46% (fig. 20a). Assume the default se-

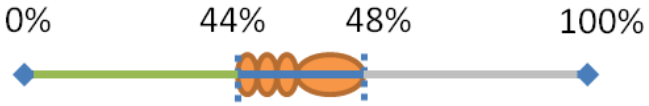
quence's next operation after reaching 44% was to expand another node cluster than the one interactively expanded called C_0, modifing the measure to 46% as well (fig. 20b). If the user now moves the slider to 48%, the heuristic that defines the order of the operations can not consider that the cluster C_0 is expanded and can not expand C_1, C_2 or C_3 anymore. We thus need to define a way to smoothly go from the configuration before and the configuration after user interactions. In this example, we need at least to redefine the default sequence between 44% and 48% (fig. 20c). But even if it seems still valid to consider the old part of the sequence between 0% and 44% and beyond 48%, nothing forbid us to change it as well (fig. 20d). In any case, it won't be possible to keep the default sequence as is for the values that are concerned by the user interactions, even below the starting value if user collapsed clusters.



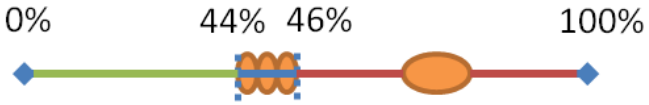
(a) Alternative sequence defined by user after the expansion of C_1, C_2 and C_3.



(b) Default sequence before interaction, next action after 44% is opening C_0.



(c) Smallest modifications to be consistent with user interactions. The part in gray can be or not identical to the one of the old default sequence.



(d) Modifications of everything beyond 46% to improve the next explorations for the user.

Figure 20: Modifications of the default sequence following user interactions.

Several solutions, all implying learning on the interactions made by the user, are feasible. We can for example use the sequence defined by the user to go from one state to the other. By doing so, we change the order of the operations by inserting the interactive ones where they happen. This solution offers only few changes and is not satisfactory because it modifies only the behavior of the default sequence for the very elements affected by the user interactions. If the user has a particular pattern in mind and sticks to it, then every time that it

is different than the default sequence, changes will be necessary. On the other hand, we can try to understand the pattern that the user is following and correct the clustering globally.
Note that in all the solutions, we have to check that the sequence is consistent with the structure of the graph and that no cluster is planned to be expanded if it is not visible (because its parent hasn't been expanded yet for example).

It seems that all the remaining questions could be solved at once. Indeed, all the solutions imply the use of learning procedures. For this reason, we will now present what to learn, how to learn it and how to use it for our problems.

5 Interactive Learning

User interactions are the best way to capture the preferences of the user and to improve the tools he has. For example, the rules of expansion for the slider can be modified if we are able to find a trend in the actions of the user. These modifications of the default sequence will also smooth the gaps between the current configuration resulting of the user operations and the old default sequence for the same proportion of information because the trend discovered in the interactions will promote these interactions in the default sequence, reducing the differences between the two sequences.
If the user interacts with the slider again, then the operations can simply be the most similar ones in the set of the remaining ones with respect to the trend detected before. To do this, we can expand first the ones that were expanded before in the default sequence but not yet in our configuration and collapse the ones that were not in the default sequence at this point.

But these assumptions have a huge requirement. We need a learning procedure that will capture user interests in this graph and we will see that this interactive learning will bring several new challenges. To achieve this, we focus on the preferences of the user in terms of interactive operations. We will try to *learn* which criteria make him expand an element rather than another. The final goal is to reuse those rules to determine the order in which operations should be made with the slider.

5.1 Simple model

Given the characteristics of a node and an edge we have several parameters to help us define the rules ordering the operations. We can consider the number of neighbors, the number of nodes or edges in the cluster, the fact that the cluster is a node or an edge, the number of parents, the kind of the parents, the size of a node, the intensity of an edge, the depth in the hierarchy, the attributes of the nodes or edges, etc. In the use-case, we are mainly using 3 parameters: the type of the cluster opened, the quantity of measure contained in the cluster (using the α and β parameters as well as the number of nodes and edges contained) and the depth of the cluster in the clustering hierarchy (1 for the root, 2 for its children, and so on).

With those parameters, we will try to catch the trend in the operations performed by the user to change the exploration path and reflect these changes in the default sequence. For this we need something to sort the possibilities in the sequence with respect to the last changes and we will associate a weight factor to each parameter that we analyze. We will thus have three weights in our use-case: one for the type of cluster, one for the quantity of measure contained in the element expanded and a last one for the depth of this element.

These weights might be modified at each step with the new constraints coming from user modifications on the graph. We will consider the different alternatives that the user had at the moment of the interaction and we will create constraints based on the preference of one interaction on the others. In the following, we will call the favorite action a_0, the other alternatives $a_1, \ldots, a_n$ and the set of actions $\mathcal{A} = \{a_0, a_1 \ldots, a_n\}$. Then, for every interaction, we will add $a_0 \succ a_1, a_0 \succ a_2, \ldots, a_0 \succ a_n$ to the constraints that can be read as "a_0 is preferred to a_i". Once we have this set of constraints, we will need to interpret the constraints as equations to relate the weights factors and the interactions. For this purpose, we will use an utility function $u : \mathcal{A} \to \mathbb{R}$ whose property is that: $\forall x, y \in \mathcal{A}, x \succ y \Leftrightarrow u(x) \geq u(y)$ and that will transform the abstract order between the actions into the concrete order over the reals associating a value to each action.

We decide to start with a very simple utility function, $u(x) = \sum_{j=1}^{p} w_j x_j$ where x_j is the j^{th} parameter of interest for the interaction $x \in \mathcal{A}$. In our case, it will give $u(a_0) = a_0^1 w_1 + a_0^2 w_2 + a_0^3 w_3$, where a_0^1 is 1 if the cluster expanded in the action a_0 is a node ans 0 otherwise, a_0^2 is the amount of information contained in this cluster and a_0^3 is the depth of the cluster. With this function, we transform the set of constraints into a set of equations of the form $\sum_{j=1}^{p} w_j a_j^0 \geq \sum_{j=1}^{p} w_j a_j^i$ equivalent to $\sum_{j=1}^{p} w_j (a_j^0 - a_j^i) \geq 0$ for $i = 1, \ldots, n$ that represents the preference of action a_0 over action a_i. These equations will be the constraints of the optimization problem to come.

While solving toy examples with the graphical representation of the constraint set, we remarked that the vector $\vec{w} = \vec{0}$ is always solution but doesn't give any information for the modification of the default sequence — this is a trivial solution. Moreover, it implies that all the constraints pass through the origin and thus there are always either an infinite number or zero non-trivial solution (fig. 21a and 21b). So we decide to select the optimum solution between all the possibilities by two similar techniques given that the number of existing solutions that solve all the equations is 0 or ∞.

Intuitively, we would like to pick the solution that is the furthest from the borders of the domain for the case where the set of solution goes to infinity and conversely when no solution exists, we would take the point that satisfies the most the constraints, the closest from the borders defined by the constraints. Mathematically, by adding an error variable in every equation, we can represent the distance to the boundary. We now consider: $\sum_{j=1}^{p} w_j (a_j^0 - a_j^i) \geq \varepsilon_i$ where ε_i is the distance to the border of constraint $a_0 \succ a_i$. With respect to the intuition we gave earlier, the optimum solution will be the one that maximizes the

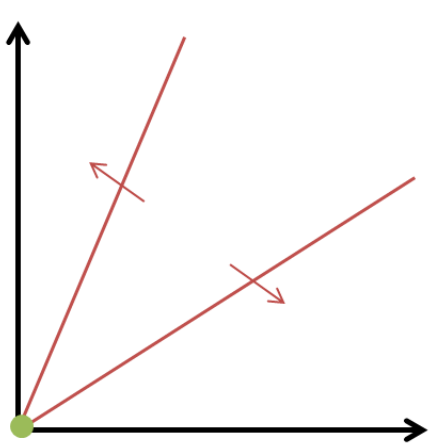
sum of all the epsilons, with $\varepsilon_i \geq 0$ if there are an infinite number of solutions and with $\varepsilon_i \leq 0$ if there is no solution.

Nonetheless, the same problem remains because even if intuitively this solution seems the best, it won't solve it alone. Indeed, the optimum solution for an infinite domain of solution will always be at infinity because it is where the distance to the border is maximized and for the case with no solution, the smallest possible error, leading to the neighborhood of the origin, will always be sufficient to solve the system but any ratio between the weights will be an optimum solution with this definition. Actually, it is only these ratios that are interesting for us because to sort the default sequence regarding a particular set of parameters, we simply need to know which one is promoted by the user and applying weights 1 and 3 will be the same as 1000 and 3000 respectively. For this reason we can normalize the weights by any mean and the given information will be enough. However, we have technology limitations here and this is why we prefer to solve linear optimization for which constraints have to be linear in the dimension of the variables (here the weights). This leads to the following normalization constraints: $\sum_{j=1}^{p} w_j = 1$ and $w_j \geq 0$.

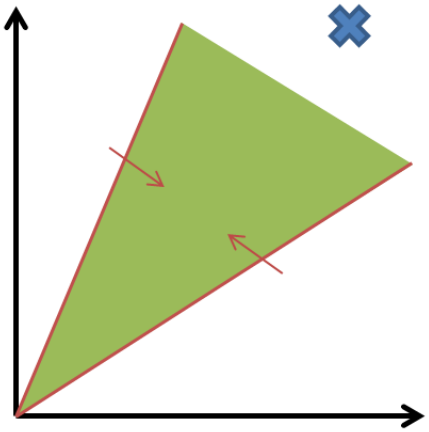
Fig. 21c and 21d show the resolution of the same constraints with the normalization constraints and the error variables ε_i added. We remark that there exist multiple solutions for both problems even with these modifications. In this case, we propose to find all the extremities of the solutions set and to compute a geometrical average to select the final solution. The extremities can be computed by maximizing and then minimizing one variable at a time in addition to the previous system of equations.

Let's now summarize all the modifications that we did to our resolution method:

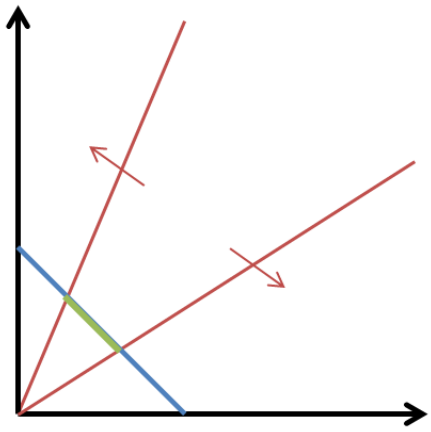
1 - Extract interesting parameters from the different possible actions;
2 - Solve the system of equations defined by the preference of one interaction over the others: $\sum_{j=1}^{p} w_j(a_j^0 - a_j^i) \geq 0$;
3 - If there is no non-trivial solution, solve the following linear optimization problem with $\varepsilon_i \leq 0$;
4 - Otherwise (if the solution set is of infinite size), solve the same following linear optimization problem with $\varepsilon_i \geq 0$;
5 - Maximize $\sum_{i=1}^{n} \varepsilon_i$ subject to $\sum_{j=1}^{p} w_j(a_j^0 - a_j^i) \geq \varepsilon_i$, $\sum_{j=1}^{p} w_j = 1$, $w_j \geq 0$;
6 - If there are more than one optimum solution, then compute the optimization that maximizes and minimizes every weight one by one and select the average point as solution.



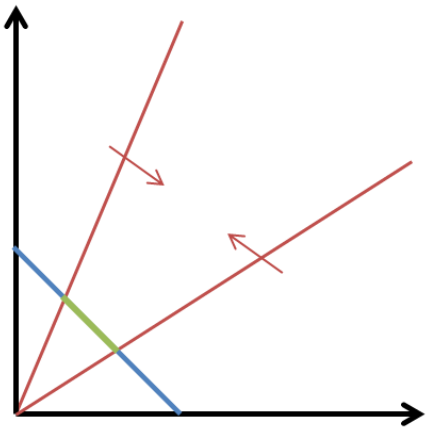
(a) A unique trivial solution.



(b) Infinite solution set with optimum at infinity.



(c) With normalization constraints, the solution is an infinite set.



(d) With normalization constraints, the solution is the same infinite set.

Figure 21: Solutions of a simple constrained example where the red lines are the constraints, the blue lines the normalization constraints, in green the solutions and the blue cross is the optimal solution. In (c) and (d), all solutions are optimal.

5.2 UTA method

With the system presented above, we will face configurations where $a_j^0 - a_j^i$ might be negative for all the parameters $j = 1, \ldots, p$ and then won't solve the normalization constraint without error. In this case, the best solution will be to put all the weight on the parameter that is the closest to zero (see fig. 22). Indeed, this parameter will be the one requiring the smallest error to have a valid system. But this won't give a solution representative of the meaning of the constraints. Remember that the problem comes from the fact that for all parameters, $a_j^0 - a_j^i < 0$. This means that the user preferred small values

for all the parameters and this will now be interpreted as the user willing to maximize the criterion he tried the least to minimize. To solve this gap in the comprehension of the interaction, we will need to interpret the negative values as minimization of the parameter and positive values as maximization.

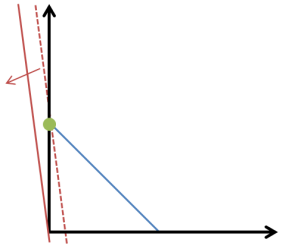
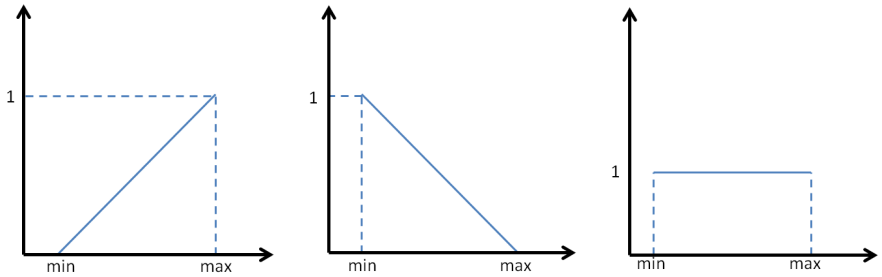


Figure 22: Graphical resolution of a constraint with negative values.

In general, this problem never occurs because we always know if our goal is to maximize or to minimize when we associate a parameter with an utility function but here the problem comes from the fact that the user determines the values to interpret and that linear optimization requires linear normalization constraint. Thus, the best solution would be to compute the optimization for all possibilities and to decide for minimization or maximization independently for all the parameters based on the solution minimizing the objective function. Unfortunately, JavaScript libraries for linear optimization are not as efficient as we could find in other programming languages (such as Matlab for example) and thus we will need to find a low complexity solution for this decision, implying to solve the least optimization problems. We decided to introduce a variable for each parameter, defining if it should be maximized or minimized. These variables are called s_j and take values -1 if the parameter needs to be minimized and $+1$ if it needs to be maximized. This will force us to predefine the objectives for the parameters given the learning and to do the optimization in two phases but we don't have a better solution given the performances of our linear solver and technology constraints.

Another problem can be illustrated by the following example. Consider the constraint $2w_1 + 10w_2 \geq \varepsilon_i$ combined with the normalization constraint $w_1 + w_2 = 1$. The optimal solution is $w_1 = 0, w_2 = 1$ but the scales of the values might be really different for the parameters. For example, we could have $x_1 \in [0, 2]$ and $x_2 \in [0, 1000]$. In this case, the difference of parameters 1 for the two possibilities would have reach its maximum unlike parameters 2 that will be a bit different compared of what it could have been. Moreover, parameters that are categorical like the type of element that was expanded (edge cluster or node cluster) have to take two different values with a scale that won't be comparable to these for the number of elements in the cluster (that might be spread) and for the depth of the element in the hierarchy (that will be really tight) at the same time. For those reasons, we have to normalize the inputs to reach a unique scale where the minimum is represented by 0 and the maximum by 1 (fig. 23). Separately for each parameter, we store the minimum and the maximum value for all the elements and we simply represent the utility for the

parameter j with a linear formula replacing the value of the parameter x_j in the old formula. This formula will depend if we want to minimize or to maximize the given parameter, solving at the same time the previous problem. For maximization, we have $g(x_j) = \frac{x_j - \min_j}{\max_j - \min_j}$ (fig. 23a) and for minimization, we have $g(x_j) = \frac{\max_j - x_j}{\max_j - \min_j} = 1 - \frac{x_j - \min_j}{\max_j - \min_j}$ (fig. 23b). So we can use the sign variables to end with an unique equation: $g(x_j) = \frac{1-s_j}{2} + s_j \frac{x_j - \min_j}{\max_j - \min_j}$.



(a) Maximization (b) Minimization (c) Constant case

Figure 23: Normalized marginal value function for the parameters of the constraints depending on the goal of the user for this parameter.

About the slack variables ε_i, we proposed also to change their purpose to reduce the two different problems, if there were no or infinitely many solutions, into one. To do this, we now associate the error variables with the actions and not with the constraints. Thus, we have now a constraint $c_i = \sum_{j=1}^{p} w_j(g(x_j^0) - g(x_j^i))$ that needs to be greater or equal to $\sigma_i - \sigma_0$ and all the slack variables are non-negative, $\sigma_i \geq 0$. The optimization problem becomes now to minimize the sum of those non-negative error variables in both situations (with no or infinitely many solutions without slack variables). However, since we are simply summing the error variables in the objective function, we often face the case where more than one solution was achieving the smallest value for the objective function. In this case, it is thus needed to apply the final step like it was the case with the simple model to determine the geometrical average of the solutions as the result. Note that since we changed the point of view of the errors from constraint errors to action errors, then the error σ_0 should be counted as many times as there are constraints since it was in every constraint error.

With these problems solved, we can construct a new system that will have to obey to the following constraints:

$$\begin{cases} \text{Minimize } n\sigma_0 + \sum_i \sigma_i \\ \text{Subject to:} \\ \sum_j w_j g_j(x_j^0) + \sigma_0 \geq \sum_j w_j g_j(x_j^i) + \sigma_i \\ \sum_j w_j = 1 \\ \sigma_i \geq 0, w_j \geq 0 \end{cases}$$

Recall that we first need to determine the signs s_j before solving it and note

that this system contains exactly the constraints of the UTA method where the principle is to add together the utility of every parameter separately. The whole theory can be found in [19] and correspond mainly to the problems faced here.

In the end, the weights used to sort the different actions of the default sequence only depended on a normalized factor and a sign. The factor is used to determine which parameter is the most important and the sign to know if the user preferred to minimize or to maximize this parameter. Thus, it would have been useful to consider the normalization criterion as the unit circle: $\sum_{j=1}^{p} w_j^2 = 1$ that would have give all the information at once and avoid the preselection of the best signs. Unfortunately, this would have transformed the problem into a non-linear optimization problem which is much more complicated to solve and prevent the operations to be real-time.

We remarked also that a constraint that appears several times can be kept only once as long as we count it correctly when deciding for the signs and in the objective function, when we sum up all the slack variables associated to the other actions. This allows us to reduce the size of the systems to optimize and thus to increase the speed of the computations.

5.3 Sign decision

If the procedure to determine the sign variables hasn't been presented above, it's because it remains an open problem for us. Nonetheless, we will present different intuitive ideas to determine the signs for the parameters before the optimization. Our main goal is once again to use low complexity algorithms to avoid increasing computation time. Indeed, to find the optimal solution, we would need to do 3^p optimizations with all the combinations of values in $\{-1, 0, 1\}^p$ and to keep the combination that minimizes the objective function. This is something we cannot afford with our technology limitations if we want to achieve real-time computations.

5.3.1 Constant parameters

Until now, we always chose to minimize or to maximize a parameter, depending if the parameters of the constraints were negative or positive, respectively implying that the selected element had a smaller (resp. greater) value than the other alternatives for this parameter. But we noticed that these parameters were also equal to zero sometimes and that this value could be the majority, mostly when the constraints came from an interaction on the minimal graph. In this case, it is hard to know if we better have to minimize or maximize this parameter.
We choose then to add a third possibility, that is to consider the *marginal value function* $g_j(x_i)$ as constant whatever is the value of the constraint to express the fact that we are not interested by this parameter because it was not significant enough in most of the constraints to be interpreted (fig. 23c). This is represented as a sign with a null value.
With this new possibility, we created solutions where the weights where not giving any information in the computation of the new default sequence. Indeed,

those solutions were giving all the weight to the parameters that had a constant normalization since it was minimizing the objective function but of course, it was not representative of the constraints.
For this reason, we had only two choices available when a parameter of the constraints is not significant enough to be interpreted. Either to choose randomly between maximizing and minimizing this parameter or to force the weight associated with it to be equal to zero when the marginal value function is constant. The first one will give no interpretation of this parameter for the constraints where the value was 0 but will give information for the potential minority being non-zero whereas the second solution will never have an impact on the learning procedure; it will simply omit this parameter in the optimization.

5.3.2 Majority detection

We proposed next to count the number of times that the constraints were positive, negative or null and to select minimization if there was a majority of negative values for this parameter among the constraints, maximization if there was a majority of positive values and not to care about this parameter or to decide randomly if there was almost the same number of positive and negative values for the parameter or if there are only zeros.

This solution seems to be representative of the expectations of the user. Indeed, if his constraints are almost all positive for one parameter, then it means that he tried to maximize this parameter. Nonetheless, the combination of simple constraints can lead to unsuspected solutions. For example, if we consider the constraints $-0.9w_1 + 0.1w_2 \geq \sigma_1 - \sigma_0$ and $0.2w_1 - 0.8w_2 \geq \sigma_2 - \sigma_0$, then there are solutions with no error if we minimize both parameters. But if we choose to maximize one or both of the parameters, then we have strictly positive slack variables. Here, this is not a problem of quantity of constraints because 100 times the first one and once the second one still give better results for minimization of the two parameters. So the majority decision would give wrong results: minimization of the first parameter and maximization of the second one.

5.3.3 Neighborhood solution

We have finally decided to use a brute force technique to have an idea of the signs for the parameters. To do so, we define a set of points in the neighborhood of the origin and we compute the distance to the constraints for those points. Then we take the signs of the point in the set that minimizes the total error (sum of the errors).
If we continue with the same example than before, we have to compute the values of the different utility functions to determine the minimum value for the error variables. To compute this, we set the weights as follows: $w_1 \in \{+0.001, -0.001\}, w_2 \in \{+0.001, -0.001\}$ and we derive the slack variables for all the possibilities. After computation, we remark that setting both weights to -0.001 allows both errors to be null and thus minimizes the objective function.

Since we had to choose between accuracy and speed to solve the problem of the detection of the signs, we preferred to introduce this solution which is more

efficient than the previous one. Even though, we hope that in the future we will be able to have a system that is both accurate and fast in order to achieve real-time learning.

5.4 Trend detection

We now have a complete model from the interactions to the modifications of the default sequence. However, we didn't discuss yet the procedure to choose when to learn and just assumed we store the parameters for all the possibilities that the user had when he collapsed or expanded an element for all the interactions. We will now develop those questions to capture as much as possible the experience of the user. We will also remark that those two questions are closely related.

Let's first come back to the question of what is stored. We started this section by assuming that everything was stored, for simplicity of presentation of the UTA method but also because we need our learning process to recall that the user promoted one action over all the other possibilities he had. This means that he is more interested by this element than by any other. Nonetheless, storing every unselected action at every step of the interaction will eventually lead to high error values in the optimization process and thus might not be representative of the goals that the user had. We will see that this problem could be solved at several levels. We could either split the interactions of the user in sessions, change the granularity in function of our point of view or we could simply learn on subsets of the constraints set defined by specific criteria. Using both is also a solution and is actually the one used for the first tests.

Concerning the session principle, we can choose several level of separation of the information. We could consider that every interaction has a different purpose and so learn after each interaction and then forget the constraints. Conversely we could consider that everything is always related to the same goal of the user even if he starts the tool again because he has one main business and behaves always in the same way when interacting with his data. As an intermediate solution, we could argue a split between the connections, where the data is stored as long as the user doesn't quit the tool. We select the last option that doesn't require local storage and will be the easiest while discussing the suppression of subsets of constraints in the following. Moreover, it is the most meaningful one because experiments showed that changing after each interaction often leads to configuration 20c and never forgetting anything leads to the huge sums of errors we are trying to avoid.

Like with the session problem, several solutions can be used to determine how to select a subset of constraints to consider in the optimization. Among all the possibilities, we focused on a few solutions that seemed to be coherent with our model. First, we tried to remove from the entire set the constraint that produces the biggest error σ_i because if this constraint is not in the set anymore, then the objective function will decrease by at least this value. We discovered that this solution is not optimal because a constraint associated with a slack variable relatively close to 0 might force other constraints to increase their slack variables in order to solve the system and thus increase a lot the value of the objective function (fig. 24). However, it gives one of the best results in

average and it is a low complexity method that doesn't require to compute the optimization for every subset of size $n-1$. Moreover, note that this idea could be extended to any size and would increase even more (and exponentially fast) the number of optimizations to run. We also need a criterion to decide when to apply this technique that reduces the size of the constraint set and how many times. We propose to apply it as long as the objective function is over a particular threshold, defined with respect to our needs of precision which require more experiments to be defined clearly.

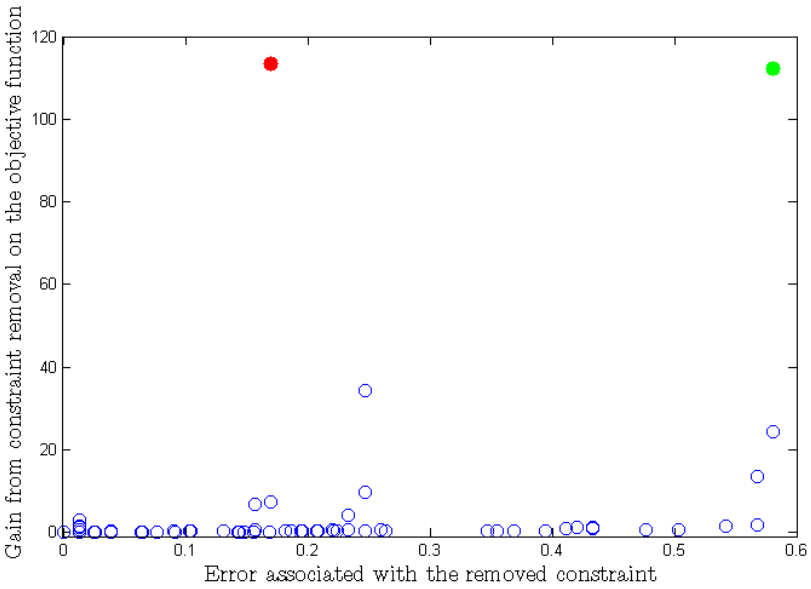


Figure 24: Test on constraint suppression that shows that the removal of the constraint with largest slack variable (in green) is one of the best solution to minimize the objective function (optimum is in red).

We also explored the fact that if constraints start to be conflicting at some point, it might be due to a change in user interests. Thus, we tried to remove the oldest constraints from the set. But this test was not improving significantly the value of the objective function in the optimization problem when we were removing the oldest or the two oldest constraints. There are several explanations for that. The first one is that even if our idea is right if we discover it at the moment where the conflicts happen then it will often be the case that there are more constraints associated with the first exploration of the user than with the second so removing the last few constraints won't solve the problem and more than half of the constraints should be deleted to obtain improvements (fig. 25). Another reason is that one interaction triggered by the user will result in several different constraints (as many constraints as there were alternatives for the user when he had selected the cluster) so the removal of a few constraints won't be significant at all.

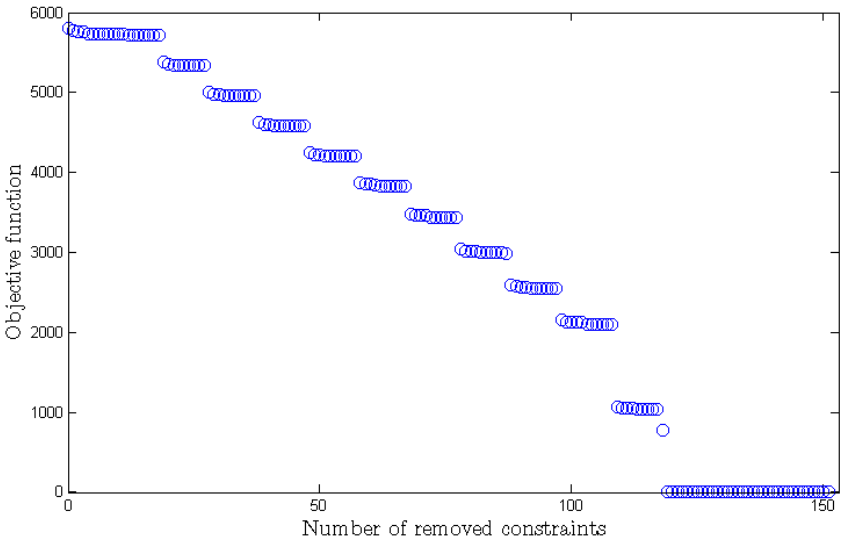


Figure 25: Evolution of the objective function when removing the constraints one by one from the oldest to the latest.

From this approach, we kept the idea that a constraint might be interesting only during a limited number of steps and that if user continues to explore with the same goals, then the same constraints should appear regularly. Following this intuition, we introduced a lifetime that we associated with the constraints. This is an hybrid solution between the session problem and the subset problem which gives interesting results (fig. 26).

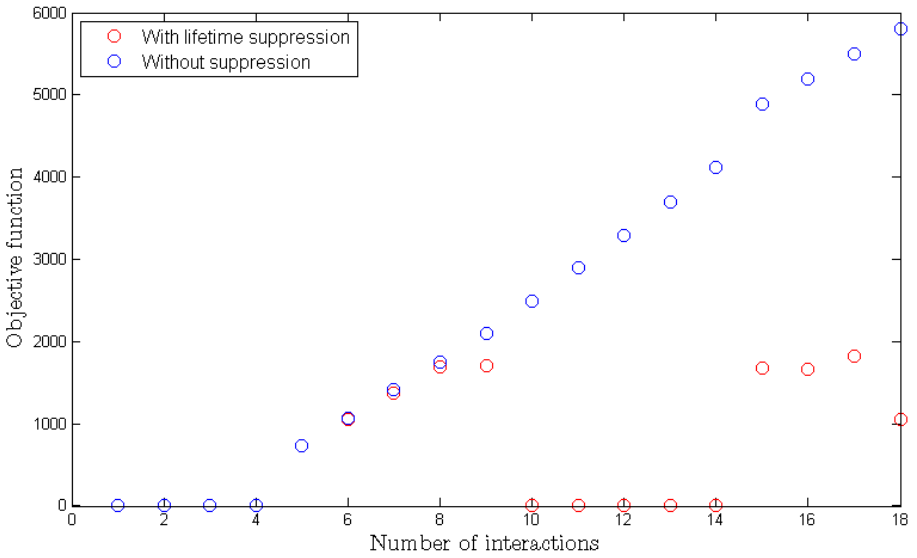


Figure 26: Evolution of the objective function when constraints are taken into account only in the 5 steps that follow their apparition.

Fig. 27 summarizes all the steps of the learning that are necessary to transform the interactions of the user into something that can be useful for his next interactions in particular with the slider. The decision for the signs in the utility function and the detection of conflicts are not optimal yet and need more tests to be efficient. However, we think that solving these challenges could significantly help the user in the exploration of his data and that a low complexity solution is the only way to keep the interactions real-time with big sets of data. Moreover, given the detection principles above, it is clear that the learning should happen each time that the slider is used to edit the default sequence just before it is needed, using as much information as possible.

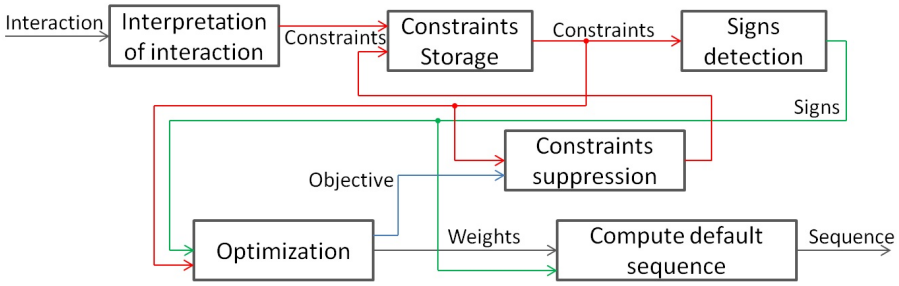


Figure 27: Diagram of the steps of learning for the modification of the default sequence.

6 Conclusion

This report presents a new tool developed to understand and improve user experience in the exploration of data represented by graphs. We focused on the concept of visual entities [5] and proposed a slider to allow the users to display the quantity of information they want. We introduced notions such as the default sequence to determine the data displayed with respect to the value of the slider that users choose. We linked in this process the simplicity of a slider with the complexity of the graph representation that led to many different solutions and for which we added a learning mechanism. This step took user interactions on the graph as information that helped to display data following his goals when he uses the slider.

We know that challenges remain in this area and that the approaches used here to solve the problems might be not optimal. For instance, we need to understand completely the issues linked to the selection of the signs for the utility functions. We know that it is important to decide if we need to minimize or maximize a parameter in the default sequence but we are lacking evidence that the majority of the sign of the constraint is a sufficient test. We think for example that some combinations of constraints could lead to resolutions that are not trivial and that could depend on several criteria at once.

We also suggest to improve the detection of the constraints to remove in a twofold way. First by defining a strong criterion for the removal of the con-

straints, if possible even without applying any optimization first to improve the runtime. Second by determining how many constraints and which ones should be deleted to get acceptable solutions in one step. Recall that in this report we presented only solutions deleting constraints one by one and applying optimization to check if we needed to continue the removal.

Another concept that might be closer to the goals of the users could be to consider several interactions in a row together. For example, during the exploration of the graph a user might be interested by data that is deep in the hierarchy and then he will need to expand several clusters without any interest for them in order to reach the data he wants. So his interactions are not relevant by themselves but as a unit. This implies to know or estimate how many interactions in a row are part of the same exploration before applying the learning part.

More generally, it would also be a good improvement to be able to mix attribute-based and structure-based clustering for learning. The problem comes from the fact that user often looks for data for its attributes and then he might need to expand and collapse a certain quantity of clusters before reaching the data he is interested in if the clustering is only structural. The major problem is that we miss techniques for attribute-based clustering that would be suitable for learning. Indeed, we will also need attribute-based parameters to learn how user explore his graph once we will have a good clustering.

Appendix

A Repartition of computations

The tool developed here is separated in two parts. One running on the server, written in Java responsible of the main computations and one running on the client machine, written in JavaScript responsible of the display. In this section we will try to understand the challenges that are linked to the repartition of the code between the front-end and the back-end given the interactions of the user.

When we try to code everything in the back-end, we face the fact that every action of the user needs to be transfered to the back-end, even if no computations are needed to keep track of the state of the graph and update potentially some variables. For instance, if the user expands a node and the back-end is not aware of this, it won't update the weights for the learning, the list of the potential clusters that can be open in the next step, neither the value of the slider that will change and so it will propose operations on the graph different than the ones user could expect.

If we try to code everything in the front-end this time, then the client will query all the data from the database once. This will result in a huge query that will take time to process, to download and to parse. Moreover, once the query is parsed, the client has to store everything locally. Given the size of the database, it might not be possible to store everything. Nonetheless, the interactions can be displayed more quickly if the code is in the front-end because there is no exchange of data between the front-end and the back-end.

For this reason, we are interested in the best threshold between the two parts. The problems that we presented so far are based on the memory limitation of the front-end and the resources required for any exchange of data between the two sides. The goal here is then to minimize the quantity of data exchanged while storing the least data in the front-end to avoid that the application crashes. For this reason, it seems to be practical to compute the most we can in the back-end and with this strategy we only transmit what to display to the front-end. But this strategy implies that we compute learning and slider parts in the back-end thus to be able to combine interactive operations with the automatized ones, since we need to send to the back-end the information about each interaction.

To avoid this, we propose to do the computations for the display (including slider and learning parts) in the front-end. With this separation, we limit as most as possible the data to be stored in the front-end as well as the number of times that a request of the user is sent to the back-end.

The solution to this problem was dedicated to the operations performed by the tool. For another kind of interactions, it might be easier to separate the functionalities a bit differently. Nonetheless, the goals to achieve will be quite similar and can even be extended to a client-server separation of the code. We could imagine that the data is stored online and that the client that want to

access it has a dedicated application on his device.

In this case, we would have to reconsider the transactions between the client and the server in order to be the only one able to read the data exchanged since it could be confidential data that is displayed with the tool. In this case again, exchanging the minimum amount of data is also a way to ensure the privacy and the security of the content.

References

[1] D. Archanbault, T. Munzner, and D. Auber. Grouseflocks: Steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, 2008.

[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[3] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E 72*, 2005.

[4] N. Elmqvist, T.-N. Do, H. Goodell, N. Henry, and J.-D. Fekete. Zame: Interactive large-scale graph visualization. In *Visualization Symposium, 2008. PacificVIS '08.*, 2008.

[5] N. Elmqvist and J.-D. Fekete. Hierarchical aggregation for information visualization: Overview, techniques and design guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 2010.

[6] S. Fortunato. Community detection in graphs. *Physics Reports*, 2010.

[7] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. In *Proc. of the National Academy of Sciences*, 2002.

[8] R. Guimerà and L. A. N. Amaral. Functional cartography of complex metabolic networks. *Nature*, 2004.

[9] J. Heer and A. Perer. Orion: A system for modeling, transformation and visualization of multidimensional heterogeneous networks. *Visual Analytics Science and Technology*, 2011.

[10] N. Henry, J.-D. Fekete, and M. J. McGuffin. Nodetrix: A hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 2007.

[11] X. Huang, P. Eades, and W. Lai. A framework of filtering, clustering and dynamic layout graphs for visualization. In *Proc. of the Twenty-eighth Australasian conference on Computer Science*, 2005.

[12] Y. Jia, J. Hoberock, M. Garland, and J. C. Hart. On the visualization of social and other scale-free networks. *IEEE Transactions on Visualization and Computer Graphics*, 2008.

[13] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 1970.

[14] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proc. of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008.

[15] Z. Liu, S. B. Navathe, and J. T. Stasko. Network-based visual analysis of tabular data. *Visual Analytics Science and Technology*, 2011.

[16] M. E. J. Newman. Detecting community structure in networks. *The European Physical Journal B*, 2004.

[17] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E 69*, 2004.

[18] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. In *Proc. of the National Academy of Sciences*, 2003.

[19] Y. Siskos, E. Grigoroudis, and N. F. Matsatsinis. UTA methods. In *Multiple criteria decision analysis: state of the art surveys*, pages 297–343. Kluwer Academic, 2005.

[20] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, 2011.

[21] M. Wattenberg. Visual exploration of multivariate graphs. In *Proc. of the SIGCHI conference on Human Factors in computing systems*, 2008.