Interactive Visualization of Large Similarity Graphs and Entity Resolution Clusters

Demonstration Paper

M. Ali Rostami, Alieh Saeedi, Eric Peukert, and Erhard Rahm
Database group and ScaDS Dresden/Leipzig, University of Leipzig
{rostami,saeedi,peukert,rahm}@informatik.uni-leipzig.de

ABSTRACT

Entity Resolution (ER) identifies semantically equivalent entities, e.g. describing the same product or customer. It is a crucial and challenging step when integrating heterogeneous (big) data sources. ER approaches typically compute a similarity graph where vertices represent entities and edges (links) connect similar entities. Different clustering algorithms can be applied on such similarity graphs to finally determine groups of matching entities. In this demonstration paper, we introduce a new interactive tool to visualize and thus help to analyze large similarity graphs and large sets of ER clusters. Users can intuitively investigate the link and cluster structure to identify potential problems such as overly large clusters, cluster overlaps or singletons that might indicate the need for repair activities on the ER result. To support large graphs, computation-intensive tasks like layouting and sampling are executed on the server side as parallel or serial processes. The demo walks through different matching and clustering tasks and allows users to interactively explore the results.

1 INTRODUCTION

In the era of big data, one of the challenging data integration tasks is to identify semantically equivalent entities (e.g., describing the same product or customer) in large and heterogeneous data sources. This task is referred to as Entity Resolution (ER) or record linkage [6]. ER typically computes a similarity graph where vertices represent entities and edges (links) connect similar entities with a pair-wise similarity above a predefined threshold. Matching entities can directly be derived from such a similarity graph or from groups of matching entities determined with subsequent clustering algorithms [9]. Our recently introduced framework FAMER (Framework for Multi-source Entity Resolution) supports a parallel computation of such similarity graphs and ER clusters for multiple (≥ 2) data sources [12].

During the continuous development of FAMER, it is difficult to investigate the correctness and efficiency of the certain algorithms and to understand the problems. Such an investigation may lead to introduce better match and cluster algorithms or even an extra postprocessing step of repair (for more details see [13]). However, to identify (or debug) such issues with limited effort and time we see the need for a comprehensive and powerful approach to visually analyze similarity graphs and ER clusterings. Unfortunately, general purpose graph visualization tools like Gephi[1] or Graphviz[2] have only limited capabilities to analyze ER clusterings and also have problems to visualize large (big data) similarity graphs with vertex and edge properties.

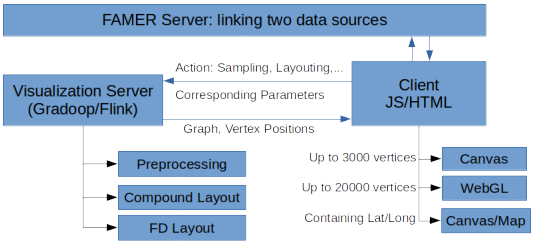

Figure 1: The software architecture

In this demo paper we introduce SIMG-VIZ, a new visualization system for entity resolution and clustering that allows us to investigate different match and clustering techniques for multi-source entity resolution. SIMG-VIZ offers the following key features:

- SIMG-VIZ allows a user to analyze precomputed similarity graphs and clusterings from existing ER tools and also supports executing and analyzing ER match tasks directly with FAMER.
- Different graph and ER cluster visualization techniques and layouts can be applied to choose the best visualizations.
- To increase performance, some layouts can be precomputed on the server with either parallel or serial computation. This provides a significant optimization potential in particular for force-directed layouts [3].
- To support visualization of large graphs, preprocessing techniques such as sampling (also executed in parallel on the server) can be selected to obtain a fast overview of large similarity graphs and their clustering results.
- Clusters and their overlaps as well as edges annotated with their type and similarity are visualized by using a simple but useful cake-like visual metaphor. Users can interact with clusters and select individual clusters for investigation.

2 SIMG-VIZ OVERVIEW

The SIMG-VIZ system consists of three modules: (1) the FAMER server, (2) a visualization server in JAVA and (3) a web-based UI-client written in JavaScript (see Fig. 1).

The FAMER server is used to link several sources and executes defined matching tasks. However, SIMG-VIZ also allows a user to load similarity graphs and clustering results that were computed by other frameworks and tools. The visualization server offers several preprocessing (e.g. for sampling) and layouting algorithms. The preprocessing algorithms are implemented in distributed fashion based on Gradoop and Apache Flink whereas

[1] https://gephi.org
[2] http://www.graphviz.org

graph layoutings are currently only implemented as non-distributed algorithms. The web-based client, provides an interactive visualization of similarity graphs and ER-clusterings. Node and edge properties can be investigated and server-side components like matching, clustering, preprocessing and layouting can be triggered. Through the client a user selects options and triggers server based REST-interfaces. The visualization server and the FAMER server both respond with JSON results. In the following paragraphs each of the components is described in detail.

2.1 FAMER Tool

FAMER offers parallel entity resolution for multiple data sources. It supports different match and clustering schemes that are executed on top of a distributed data processing engine (Apache Flink [4]) and the graph analytics framework Gradoop [10]. Its execution has two main phases: (1) computation of a similarity graph based on pairwise matching and (2) clustering. The first phase consists of several steps, namely blocking, pairwise comparisons, and match classification. Blocking reduces the number of necessary comparisons which otherwise would require to compare each entity of a data source to all entities of any other source. Different blocking techniques can be applied in FAMER such as Standard Blocking (SB), Sorted Neighborhood as well as single- and multi-pass blocking. In the matching step all entities of the computed blocks are compared by computing and combining similarities between their attributes, e.g. based on similarity measures such as Edit-Distance or Jaro Winkler. For matching, match rules can specify the required minimal similarity for the considered attribute-comparisons. The output of this step is the set of matching entity pairs (links) together with a combined similarity value per link. In the following match classification step entity pairs are classified as match or no-match based on the computed similarity values. This output is stored as a similarity graph where entities are represented as vertices and match links as edges. The clustering phase of FAMER aims at grouping together all matching vertices of the similarity graph based on the link structure and similarity values. Clustering algorithms typically try to maximize the similarity between entities within a cluster while the similarity between entities of different clusters should be minimized. FAMER supports several clustering algorithms for ER such as Connected Components, Correlation Clustering (CCPivot) [2, 5], Center [9], Merge Center [1], and Star [1].

With SIMG-VIZ, all components of FAMER can be configured and parameterized which allows users to run and compare different ER match tasks.

2.2 Client (Web-based HTML/JS Frontend)

Fig. 2 shows an overview of the web-based client of SIMG-VIZ. At the top part of the client, several options can be selected. The user can choose between different clustering and match configurations which are stored after executing FAMER. Before visualizing the similarity graph some available preprocessing algorithms such as sampling can be applied. Moreover, the user can select layouting options, i.e. which layout to use and where the computation should take place. In particular for large graphs, computing the layout on the server gives significant improvements. Finally, SIMG-VIZ offers a list of actions (see Table 1) that support different drawing tasks of the graph and some statistics computations can be triggered. On the right side of the client a number of parameters for visualization can be set. To improve interactivity, styling tasks are performed on the client, for example changing vertex or edge sizes.

Table 1: Actions in SIMG-VIZ

Action	Description
Draw graph (Cytoscape)	Draws a sim-graph in Cytoscape[3].
Draw graph (WebGL)	Draws a sim-graph in WebGL based on VivaGraph.
Compute only	Executes preprocessing and sampling without visualization.
Compute labels/keys	Computes all labels and property-keys of the vertices and edges for filtering in the left part of the UI.
Save as image	Exports an image of the drawn graph.
Remove selected node	Removes a selected node.
Degree Distribution	Computes the degree distribution of the graph.
Graph Statistics	Computes additional basic statistics of a graph.

2.3 Visualization Server

The visualization server offers preprocessing and layouting services. The preprocessing algorithms are implemented in a distributed fashion based on Gradoop and Apache Flink. Table 2 lists currently implemented preprocessing components. All layouting algorithms are available both for the client and the server as non-distributed algorithms. We observed that executing layouting algorithms that need iterations like the force directed layout [8] should not be executed on the client within a browser. Executing it on a server brings significant run-time improvements, even without distributing the computation to multiple nodes. After a layout computation finishes on the server, the positions of nodes are send to the client together with the graph. As future work we plan to implement parallel versions of layouting to be run on top of Flink or Gradoop.

Table 2: Preprocessing algorithms in SIMG-VIZ

Preprocessing	Description
Graph sampling	Computes a statistical sampling of a graph. Currently SIMG-VIZ implements vertex, edge and page rank sampling.
Graph summary	Computes a graph summary by grouping dense subgraphs of a graph to generate a compact overview of a large graph [11]. In SIMG-VIZ the Flink implementation is used.
Cluster neighbor filtering	In particular for ER-Clustering a filtering to neighbors of clusters is needed. The user specifies a cluster ID (at the right part of UI) and that cluster together with its neighbor clusters is visualized.
Cluster sizes filtering	Only the clusters with specific sizes are visualized.
Cluster Aggregation	It visualizes a graph in which the cluster vertices are grouped together.

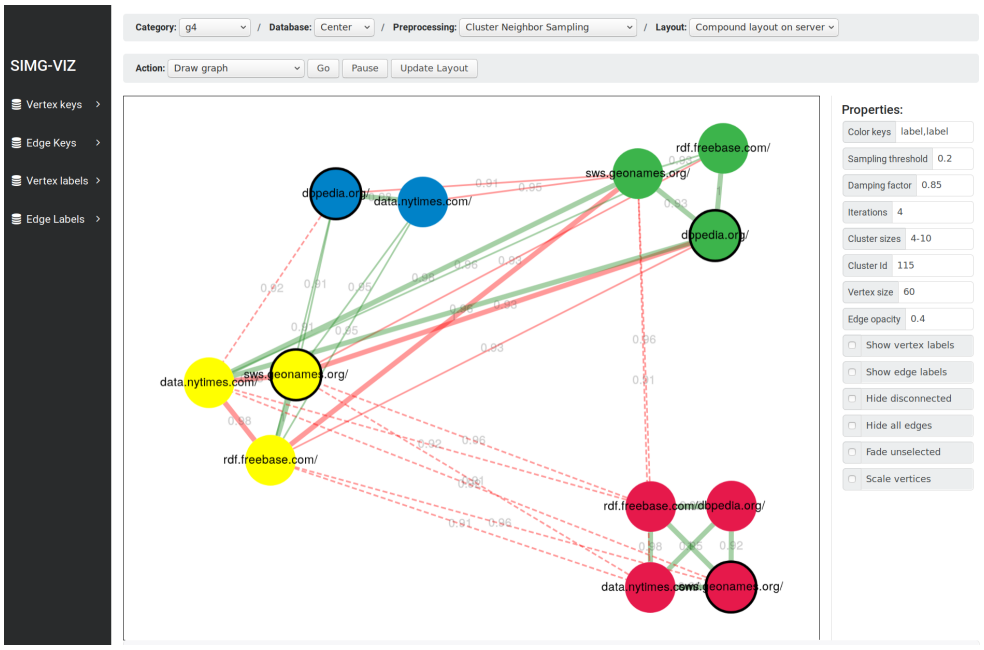

Figure 2: An overview of SIMG-VIZ

2.4 Cluster Visualization

In this section, we describe some specific features of SIMG-VIZ which are designed particularly for the visualization of ER clusters.

These features are explained along a real world ER-example of integrating four duplicate-free data sources namely Freebase (Fb), New York Times (nyt), DBpedia (db), and Geonames (geo). We initially compute a similarity graph and apply different clustering techniques with FAMER. A result cluster includes only vertices from these four sources which are most probably the same real world entities. An edge connects two vertices which have a high value of computed similarity measure. In Fig. 3 we initially visualize the complete clustering result. Clusters are given different colors to indicates cluster membership. Users can identify clusters that may warrant a closer inspection, e.g., clusters with more than 4 vertices or singleton clusters. A user can zoom in and inspect the properties of each vertex. Since generic graph layouting algorithms often have problems in visualizing large similarity graphs (e.g., problem of edge cluttering) we applied a compound layout for cluster visualization. Such compound layouts of graphs like CoSE-Bilkent [7] visualize vertices in a cluster (referred to as compound in the paper) close to each other while the whole graph is visualized by using a modified force-directed algorithm. Fig. 4(middle) shows a visualization of such a compound layout which we also compute on the server side. Vertices of a cluster share the same color and are closely grouped together to form a compound.

To get a cleaner picture, a user can interactively select a specific cluster or enter a cluster ID to only visualize a specific cluster for closer inspection. Often we also need to visualize a cluster together with its neighboring clusters (see Fig. 4 (top)). The vertex labels here refer to the corresponding data source for that entity. The scenario illustrates a problem case since there are more than four cluster members with some data sources having two entities in the same cluster which should not be possible for duplicate-free sources. There is also a singleton cluster that might have

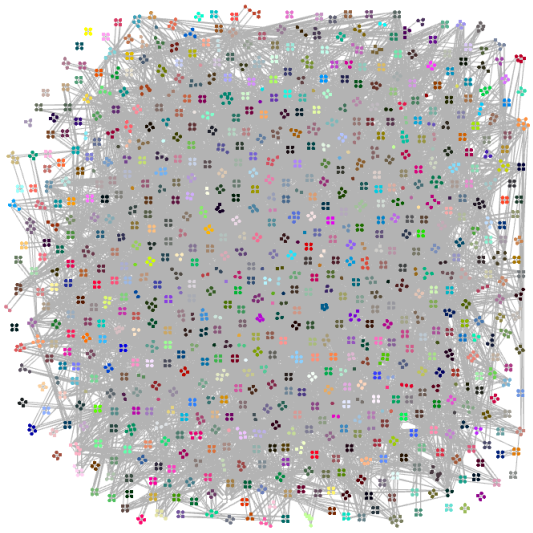

Figure 3: A visualization of all clusters.

to be merged with another cluster. Based on such observations we are now able to re-assess the used cluster algorithms and investigate new approaches for cluster repair.

We also provide support for visualizing clustering result of specific clustering algorithms like Star[9]. The Star clustering computes cluster representatives and all neighbors of those representatives are assigned to the corresponding cluster. In SIMG-VIZ those cluster representatives are highlighted with a black outline (see Figure 4). It happens that a vertex is a neighbor of several cluster representatives so that such vertex will belong to multiple clusters. These multi-assignments are represented as pie-charts on nodes. Each piece of a pie chart which has a specific color specifies a cluster assignment. For example, Fig. 4 (Middle) contains a

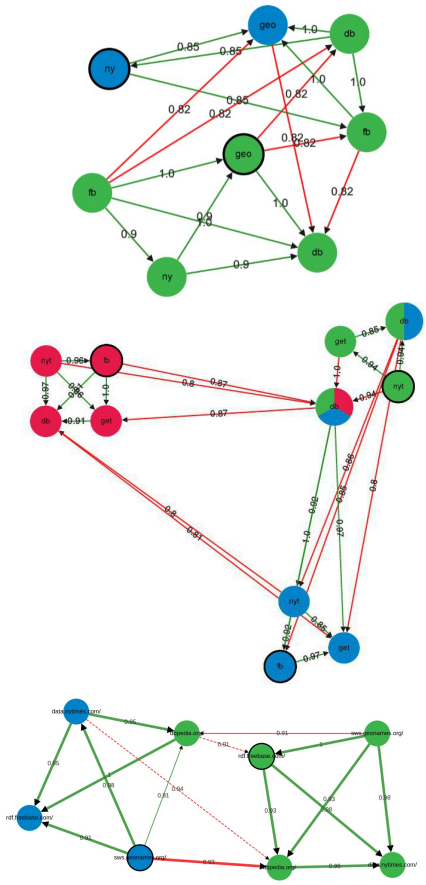

Figure 4: visualizations of (top) clusters with different colors, (Middle) vertices can belong to more than one cluster - drawn in compound layout, (Bottom) different styles for edges indicating how strong connections between entites are.

pie chart with three pieces which means that node (entitiy) is assigned to three clusters. Obviously some cluster post-processing is needed to select the best cluster for each entity that have been assigned to several clusters.

SIMG-VIZ provides special visualization support for evaluating clusters when the perfect cluster result is available for comparison. As shown in Fig. 4 (Bottom) there are different edge colors: green for edges within correct clusters and red for (wrong) edges between such clusters. Hence, clusters containing red edges should be investigated more closely. Finally we implemented a map-visualization of geo-referenced data so that entities can be plotted onto a map. With the help of this map-visualization we are able to identify false matches within a given data set.

2.5 Web-based Visualization Libraries

Drawing large graphs within a browser is problematic. We investigated several different Javascript-based visualization libraries and observed that there are significant performance differences. Three groups of libraries can be found: (1) SVG-based libraries compute SVG-nodes and tags. They are often feature rich but do not scale well due to many generated SVG-Elements. (2) The second group relies on HTML-Canvas. These libraries are typically faster but interactivity is harder to realize. Still they mostly do not scale well for larger graphs. (3) WebGL-based libraries offer the best scalability but are still not as feature rich as existing libraries in the other two categories. We finally decided to use the Canvas-based Cytoscape[4] library for small graphs up to 2000 vertices since it gives more flexibility regarding the style of edges and vertices. For example, the feature of drawing a pie chart on vertices is already available in Cytoscape. For large graphs, we use VivaGraph[5], which is a WebGL-based library with less support for vertex and edge attributes, styling and coloring. However for larger graphs the user won't be able to see those details anyway.

3 DEMONSTRATION

In the demonstration, we walk through a complete workflow of entity resolution using matching and clustering for small and large ER match problems. A user could select select data sources and the corresponding properties, similarity measures and match classifiers that are used by FAMER. The resulting clusters from FAMER are loaded into the visualization server. We then allow a user to compare different preprocessing algorithms as well as different layoutings. We consider small data sources as well as large ones.

ACKNOWLEDGEMENT

This work was partly funded by the German Federal Ministry of Education and Research within the projects Competence Center for Scalable Data Services and Solutions (ScaDS) Dresden/Leipzig (BMBF 01IS14014B) and BIGGR (BMBF 01IS16030B).

REFERENCES

[1] J.A. Aslam, E. Pelekhov, and D. Rus. 2006. *The Star Clustering Algorithm for Information Organization.* Springer.

[2] Ni. Bansal, A. Blum, and S. Chawla. 2004. Correlation Clustering. *Machine Learning* 56, 1 (2004), 89–113.

[3] G.D. Battista, P. Eades, R. Tamassia, and I.G. Tollis. 1998. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall.

[4] P. Carbone, . Katsifodimos, S. Ewen, V. Markl, S.Haridi, and Ko. Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin Issues* 38, 4 (2015).

[5] F. Chierichetti, N. Dalvi, and R. Kumar. 2014. Correlation Clustering in MapReduce. In *Proc. 20th ACM SIGKDD.* 641–650.

[6] P. Christen. 2012. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer.

[7] U. Dogrusoz, E. Giral, A. Cetintas, A. Civril, and E. Demir. 2009. A layout algorithm for undirected compound graphs. *Information Sciences* 179, 7 (2009), 980–994.

[8] Thomas M. J. Fruchterman and Edward M. Reingold. 1991. Graph Drawing by Force-directed Placement. *Softw. Pract. Exper.* 21, 11 (Nov. 1991), 1129–1164. https://doi.org/10.1002/spe.4380211102

[9] O. Hassanzadeh, F. Chiang, R.J. Miller, and H.C. Lee. 2009. Framework for Evaluating Clustering Algorithms in Duplicate Detection. *PVLDB* 2, 1 (2009), 1282–1293.

[10] M. Junghanns, A. Petermann, N. Teichmann, K. Gómez, and E. Rahm. 2016. Analyzing Extended Property Graphs with Apache Flink. In *Proc. SIGMOD Workshop on Network Data Analytics.*

[11] M. Junghanns, A. Petermann, N. Teichmann, and E. Rahm. 2017. The Big Picture: Understanding large-scale graphs using Graph Grouping with GRADOOP. In *Proc. BTW.*

[12] A. Saeedi, E. Peukert, and E. Rahm. 2017. *Comparative Evaluation of Distributed Clustering Schemes for Multi-source Entity Resolution.* Springer LNCS, 278–293.

[13] A. Saeedi, E. Peukert, and E. Rahm. 2018. Using Link Features for Entity Clustering in Knowledge Graphs. In *Extended Semantic Web Conference.* submitted for publication.

[4] http://js.cytoscape.org

[5] https://github.com/anvaka/VivaGraphJS