# Summarization meets Visualization on Online Social Networks

**Hans-Henning Gabriel***, **Myra Spiliopoulou***, **Emmanouela Stachtiari** †, **Athina Vakali**†
**Faculty of Computer Science Otto-von-Guericke-University Magdeburg, Germany*
*Email:* {*hgabriel, myra*}*@iti.cs.uni-magdeburg.de*
†*Faculty of Computer Science Aristotle University Thessaloniki, Greece*
*Email:* {*emmastac, avakali*}*@csd.auth.gr*

***Abstract*—Getting an overview of a large online social network and deciding which communities to join is a challenging task for a new user. We propose a method that maps a large network into a smaller graph with two kinds of nodes: a node of the first kind is representative of a community; a node of the second kind is neighbor to a representative *and* reflects the semantics of that community. Our approach encompasses a learning and ranking algorithm that derives this smaller graph from the original one, and a visualization algorithm that returns a graph layout to the observer. We report on our results on inspecting the network of a folksonomy.**

***Keywords*-social network visualization, social network summarization, community representatives, social networks, communities, clustering, visualization**

## I. Introduction

A newcomer in a social network needs an *overview* of the network that answers questions like: what communities are there, what are their representative actors and activities? Building such an overview is challenging: (a) communities overlap – prominent objects may belong to more than one; (b) communities are too rich to be represented by a single object, but (c) displaying all objects overpopulates the graph layout, so a summarization is necessary that (d) takes the information contributed by each graph object into account.

We propose an solution for these issues: we map a social network into a smaller yet informative graph, taking into account that some objects (like texts) yield more information than others (like tags). We then visualize this graph. Our graph summarization method appears in section III, after discussing related work in section II. Section IV contains our visualization method. We have evaluated our approach on a social tagging system, and report on the results in section V. We conclude in section VI with a summary and outlook.

## II. Related Work

The problem of mapping a graph into a smaller one is studied with graph compression methods [1]. Such algorithms replace multiple nodes of the original graph with a smaller number of *virtual* nodes. Our intention is rather to build a smaller graph of *original, representative* nodes. The discovery of such nodes is pursued by graph clustering algorithms [2], [3] that learn *core* nodes (which have many neighbors) and then use them as seeds of clusters. However, a core object is not adequate for describing a whole community (issue b in Section I). ContexTour [4] extracts and displays representative objects and a selection of their neighbors (issues b, c), but this selection is intended to visualize community evolution, rather than inform about the community semantics (issue d). This holds also for CrossSense [5], which depicts representative objects and objects frequently linked to them. CrossSense takes account of issue (a) on overlapping communities when selecting objects, but does not consider the information content of the objects. Our graph summarizer builds upon CrossSense.

## III. Identifying Representatives and their Peers

Our graph summarization method builds upon CrossSense [5], which models the social graph as a tensor. Each combination of tensor modes (users+tags, resources+users etc) constitutes a *perspective*. As in [9], CrossSense applies a generalization of PCA for higher order tensors and then uses the projection matrices and the core tensor to derive a clustering: each column of a projection matrix stands for a cluster, whose members are the objects with high absolute energy values in this column. CrossSense builds one clustering per perspective: a *pivot* object is then a top-(absolute-)energy object appearing in clusters of more than one perspective; its *world* consists of (high energy) objects that co-occur with it in more than one cluster [5]. We use the pivots and their worlds to summarize the network into an even smaller set of objects[1] which, taken together, are maximally informative of the network's clusters.

### A. Modeling network nodes as information bearers

The social network has different types of nodes (users, tags, documents etc) - these are the *modes* of the tensor. Let $x$ be an object of some mode having schema $S$, as ordered list of attributes. For $a \in S$, $x.a$ denotes the value of $x$ for $a$. The *data content* of $x$ is the concatenation of bytes of the attribute values of $x$, while the *schema content* of $x$ is the concatenation of bytes of the attribute names in $S$:

$$data(x) = concat_{a\in S}(bytes(x.a)) \quad (1)$$
$$schema(x) = concat_{a\in S}(bytes(a)) \quad (2)$$

[1]We use the terms *node* of the network, *object* in a community, and *element* of a set corresponding to a community interchangeably

where the order of attributes in the schema determines the sequencing of attributes and values.

We model the *joint information content* delivered by an ordered list of nodes $X$ as the result of compression:

$$i(X) = zip(zip_{x\in X}(schema(x)), zip_{x\in X}(data(x))) \quad (3)$$

We use Eq. 3 to find the most informative peers of a pivot $x$, choosing nodes from its *world* returned by CrossSense. We sort $world_x$ on absolute energy value, then build $peers_x$ iteratively: we first choose the top-element of $world_x$; at each further iteration, we add the node that maximizes the joint information content with respect to the elements already in $peers_x$. The node chosen at iteration $i$ is:

$$z \equiv y_i = arg \max_{y\in world_x - peers_x} (i(< peers_x, y >)$$

We call this iterative process SelectPeers. To terminate it, we can set an upper limit to the number of peers per pivot, $\tau_{peers}$, or a threshold to the size difference of the compressed outputs of two consecutive iterations. We use the former: it is easier for an observer to specify how many objects s/he is willing to inspect, than a compression threshold.

## B. Selecting representatives with their peers

SelectPeers chooses neighbors of a pivot in such a way that the joint information content (cf. Eq. 3) contributed by the pivot and its peers is maximized. However, two pivots may have common neighbors, and add no further information. Hence, we choose from the pivots a set of *representatives* for the network's communities, while maximizing *joint* information content. Algorithm 1 (Representatives&Peers) takes as input all pivots and their worlds from CrossSense, and the numbers of peers $\tau_{peers}$ and representatives $\tau_r$ that the observer is willing to inspect. If the number of communities is known, $\tau_r$ should be no less than that.

Representatives&Peers first invokes SelectPeers to choose the peers of each pivot (lines 2-4). Then, the pivot with highest energy is added (together with its peers) to the output (line 5). A further pivot $x$ is selected as representative, iff the joint information content contributed by the pivot *and* its peers (i.e. by the set $all_x$, line 4) is maximal (lines 7-9). The algorithm terminates after choosing $\tau_r$ pivots. The output consists of pairs – each representative with its peers – that summarize the original graph. Each such pair constitutes a star for a community with the representative as center.

## IV. Visualization

The input to our visualizer consists of the representatives, their *interestingness scores* for the original graph, and their peers. Interestingness is computed during the pivot identification process [5]: CrossSense partitions the network across each *perspective* into a clustering, counts the clusterings to which each pivot candidate $x$ belongs ($n_x$), computes the largest number of clusters containing $x$ within a clustering

**Algorithm 1:** Representatives&Peers

```
Input  : social network $G$, number of pivots $np$, size of a world $sw$,
         number of representatives $\tau_r$, number of peers $\tau_{peers}$
Output: set of representatives $R$, each $x \in R$ associated with $peers_x$
1  $CrossSense(np, sw, P)$ // Returns pivots $P$ with their worlds
2  foreach $x \in P$ do
3      $peers_x = SelectPeers(x, world_x, \tau_{peers})$
4      $all_x \leftarrow \{x\} \cup peers_x)$
5  $x \leftarrow$ top object in $P$; $R \leftarrow \{x\}$; Output $\leftarrow \{x, peers_x\}$
6  $Q \leftarrow all_x$
7  while $|R| < \tau_r$ do
8      $y \leftarrow arg\max_{x\in P\setminus R}(i(< Q, all_x >))$
9      add $y$ to $R$; add $all_y$ to $Q$; add $(y, peers_y)$ to Output
10 return Output
```

($N_x$), and sets the interestingness score to $\frac{n_x}{N_x}$. $x$ is interesting if $n_x$ is large and $N_x$ is small, i.e. $x$ is prominent in many or all perspectives but appears in few clusters, i.e. has an unambiguous role inside each perspective.

### A. Graph generation

If our visualizer considered only (representative,peer)-pairs as edges, it would place all peers in a circle around the representative. To draw a more compact layout, we propagate the interestingness score of each representative to its peers, also considering that a peer may be associated with many representatives. Hence, let $R = \{x_k | k = 1 \ldots \tau_r\}$ be the set of representatives, with $peers_k \equiv peers_{x_k}$. We associate each peer $v_i \in \cup_{k=1}^{\tau_r} peers_k$ with the inverse of the interestingness score of the representatives that have $v_i$ among their peers: we compute for $v_i$ a vector of weights $\mathbf{m}^i$, where $m_k^i = \frac{1}{in_k}$ if $v_i \in peers_k$, and $m_k^i = 0$ otherwise. Then, we compute the cosine similarity of the vectors and draw additional edges among similar vectors [2].

After computing the weight vectors for all peers, we generate the graph $G < V, E >$. We first add the representatives and their peers to $V$ and the edges between a representative and each of its peers. Next, we generate artificial edges between vertices (peers) that have similar vectors: we sort pairs of vertices on descending similarity, and iteratively add into $E$ the pair (an edge is a pair of vertices) with the highest similarity. This iterative process stops when $E$ contains $\frac{p}{100} \times \frac{|V|\times(|V|-1)}{2}$ edges for a threshold $p$, i.e. $p\%$ of the edges $G$ would have if it were dense.

### B. Computing the graph layout

To compute the layout $L$, we use a modified version of the Fruchterman-Reingold algorithm (FR) [6]. FR computes *attractive* forces to draw proximal nodes closely, and *repulsive* forces to prevent them from being drawn at the same location, iteratively recomputing the nodes' positions. Our modified FR also considers the similarity of the nodes' vectors when computing the forces.

[2]The computation of additional edges among similar vectors is also used in [4], but ContexTour vectors have different semantics.

The original FR uses simulated annealing, but is known to get easily trapped into local minima. To mitigate this, we start with a layout generated by the Kamada-Kawai (KK) algorithm [7] and containing nodes with low forces. From this initial layout, we expect that the modified FR will move towards local minima that are close to the global one [6].

For layout computation we use the implementation of KK algorithm, and a modified implementation of the grid variant of the FR algorithm in the JUNG 2.0 lib rary. First, we compute an initial layout $L_0$ using the KK algorithm and then apply FR on $L_0$. At each subsequent iteration, we measure the repulsion on an edge $< v_i, v_j >$ as $F_r \leftarrow f^2 \times \frac{distance(v_i,v_j)}{sim(v_i,v_j)}$, and the attraction as $F_a \leftarrow \frac{distance(v_i,v_j) \times sim(v_i,v_j)}{f^2}$, where $f$ is the force constant. The settings of the parameters $f$, $T$ (number of iterations for KK), $t_0$ (initial temperature for FR), and $maxIt$ (maximum number of iterations for FR) are as suggested in [7], [6].

The complexity for an iteration of the grid variant of FR is $O(|V|+|E|)$, for KK it is $O((1+T)|V|)$ and for similarity computation $O(\tau_r \frac{|V|(|V|-1)}{2})$.

## V. Evaluation

We evaluated our method on the quality of the graph $G$ it builds: ideally, $G$ should consist of disconnected stars. Our quality function returns the averaged portion of non-overlapping peers among the stars (larger values are better):

$$q(G) = \frac{\sum_x \sum_{y \neq x} \frac{|peers_x \cup peers_y \setminus peers_x \cap peers_y|}{|peers_x \cup peers_y|}}{\frac{\tau_r \times (\tau_r - 1)}{2}} \quad (4)$$

We evaluated on data from Bibsonomy[3], where users upload bibtex entries and tag them. A posting has the form (userId, tagId, resourceId), i.e. uploads are not observable, but tag assignments are. We used the dataset of [8], as prepared in [5]: from the 335,789 postings of 2007, the authors removed documents or tags with less than 4 postings, and postings with tags that occur more than 100 times.

### A. Summarizing a Graph around Representative Users

We derive a graph of documents and tags around representative *users*. We invoke CrossSense, varying the number of pivots $np$ for fixed max world size $sw = 30$. We then invoke Representatives&Peers, varying $\tau_r$ and $\tau_{peers}$. The baseline is CrossSense, instructed to return $\tau_r$ pivot *users* and extract the top-energy $\tau_{peers}$ objects from their worlds.

Table I depicts quality and execution time for CrossSense and for our algorithm. The improvement in quality is substantial (column 4 vs 6), and increases with the number of pivots (column 3). The overhead is one order of magnitude lower than CrossSense execution time and thus negligible [4].

[3]htt://www.bibsonomy.org

[4]Execution times are not comparable, because CrossSense is implemented in MATLAB$^R$ and our algorithm in Java. If the code were optimized, the overhead would have been even lower.

| $\tau_r$ | $\tau_{peers}$ | $np$ | CrossSense | | Representatives &Peers | |
|---|---|---|---|---|---|---|
| | | | $q()$ | Time | $q()$ | Time |
| 10 | 10 | 15 vs 20 | 0.77 | 11.3 | 0.85 vs 0.95 | 0.96 vs 0.58 |
| 10 | 15 | 15 vs 20 | 0.71 | 9.9 | 0.84 vs 0.93 | 1.33 vs 0.9 |
| 15 | 10 | 20 | 0.71 | 7.2 | 0.95 | 0.62 |
| 15 | 15 | 20 | 0.74 | 10.7 | 0.94 | 0.87 |

Table I: Performance results for each method as the number of representatives and the set of their peers grow for different $np$ values and for $sw = 30$; larger quality values are better

### B. Summarizing a Graph of Already Identified Communities

In this experiment, we apply our method on OTA (Offline Tensor Analysis) [9], which returns communities. We instruct OTA to return $\tau_r$ communities and the $sw = 30 + 1$ objects of highest absolute energy per community. The top-energy object becomes the representative of the community. Our method then selects peers from the remaining $sw$ ones; we denote this variant as ”PeersONLY”.

We compare PeersONLY to OTA and to ”TensClust”, a reference method that selects per community the $\tau_{peers}$ top-energy objects among the $sw$ objects returned by OTA. TensClust is expected to achieve best separation among the communities. In Table II, we see that PeersONLY is superior to OTA and even comparable or slighlty better than TensClust (cf. rightmost two columns).

| $\tau_r$ | $\tau_{peers}$ | OTA on $sw = 30$ peers | TensClust | PeersONLY |
|---|---|---|---|---|
| 10 | 10 vs 15 | 0.753 | 0.95 vs 0.94 | 0.95 vs 0.95 |
| 15 | 10 vs 15 | 0.751 | 0.93 vs 0.91 | 0.96 vs 0.95 |
| 15 | 15 | | 0.91 | 0.95 |

Table II: Quality of PeersONLY and TensClust for varying $\tau_r$ and $\tau_{peers}$; the quality of OTA is measured on the input it delivers to PeersONLY and TensClust; larger quality values are better

### C. Testing the Visualization Algorithm

We study the layout produced by our visualizer and check for overlapping nodes. We visualize the output of CrossSense directly, because the summarized graph of CrossSense is more challenging: it has more overlaps. We set $np = 6$ pivots and max world size $sw = 30$. We compare with the layouts of Kamada-Kawai for $T = 3000$ (Figure 1a) and Fruchtermann-Reingold for $t_0 = 100$ (Figure 1b). Our algorithm (Figure 1c) converged after 233 iterations and produced a more comprehensible layout: on Figure 1(a), we see many dissimilar nodes close to each other; on Figure 1(b), there is less overlap but there are still similar objects put apart (like 4707, 4704), and dissimilar objects put together (like 4711,150). On Figure 1(c), our algorithm draws similar objects closer and lets groups become visible, including the pairs misplaced by FR.

Finally, we verified that our method does draw similar objects together: we coupled it to a classifier and predicted

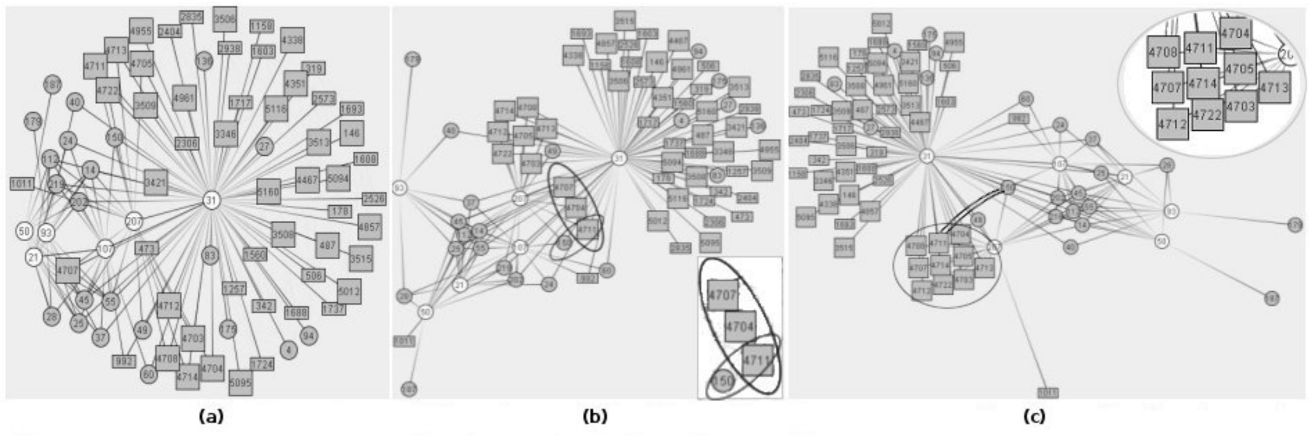

Figure 1: Visualization using (a) Kamada-Kawai, (b) Fruchterman-Reingold and (c) our new method that builds upon them

the worlds. We used the coordinates in the generated layout as features of the nodes, and the worlds as class labels. We used a k-NN classifier with k=1 to classify all nodes. The average accuracy for $sw = 6, 10, 15$ was 92.8%, 92.6% and 90.6%, respectively, i.e. more than 90% of the nodes belonged to the same worlds as the nodes drawn near them.

## VI. Conclusions

Getting an overview of a large social network involves reducing the original graph to a smaller one that still reflects the original social constellations, depicts representatives and highlights the semantics of existing communities. Our solution encompasses graph summarization and visualization. Our graph summarization method returns groups of objects that jointly contain maximal information. These groups constitute stars around representative objects. Our visualization method strives to build a comprehensible layout for them.

We have run experiments on the folksonomy Bibsonomy. We have shown that our graph summarization performs well, both when applied upon many fine-grained groups and when applied upon conventional communities. We have also shown that the visualization method returns more comprehensible graphs than the baselines. Next, we want to extend our approach to capture *evolving* networks.

## Acknowledgment

We are indebted to Andre Gohr for giving us the Bibsonomy data collected for [8].

The work of Emmanuela Stachtiari took place during her stay at the Otto-von-Guericke-University Magdeburg, on Erasmus student mobility grant.

## References

[1] G. Buehrer and K. Chellapilla, “A scalable pattern mining approach to web graph compression with communities,” in *Proc. of Int. Conf. on Web Search and Web Data Mining (WSDM’08)*. Palo Alto, CA: ACM, 2008, pp. 95–106.

[2] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, “SCAN: A structural clustering algorithm for networks,” in *Proc. of 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD’07)*, New York, NY, Aug. 2007, pp. 824–833.

[3] L. V. Thang, C. A. Kulikowski, and I. B. Muchnik, “Coring method for clustering a graph,” in *Proc. of 19th IAPR Int. Conf. on Pattern Recognition (ICPR’08)*, Tampa, FL, 2008.

[4] Y.-R. Lin, J. Sun, N. Cao, and S. Liu, “ContexTour: Contextual contour visual analysis on dynamic multi-relational clustering,” in *Proc. of SIAM Data Mining Conf.*, Apr. 2010, pp. 418–429.

[5] H.-H. Gabriel, M. Spiliopoulou, and A. Nanopoulos, “Crosssense: Sensemaking in a folksonomy with cross-modal clustering over content and user activities,” in *Int. Conf. on Knowledge Discovery and Information Retrieval (KDIR’10)*, Valencia, Spain, Oct. 2010.

[6] T. M. J. Fruchterman and E. M. Reingold, “Graph drawing by force-directed placement,” *Softw. Pract. Exper.*, vol. 21, no. 11, pp. 1129–1164, Nov. 1991.

[7] T. Kamada and S. Kawai, “An algorithm for drawing general undirected graphs,” *Inf. Process. Lett.*, vol. 31, no. 1, pp. 7–15, April 1989.

[8] A. Gohr, A. Hinneburg, R. Schult, and M. Spiliopoulou, “Topic evolution in a stream of documents,” in *SIAM Data Mining Conf.*, Reno, CA, Apr.-May 2009, pp. 378–385.

[9] J. Sun, D. Tao, and C. Faloutsos, “Beyond streams and graphs: dynamic tensor analysis,” in *Proc. of 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD’06)*. Philadelphia, PA, USA: ACM, 2006, pp. 374–383.