Analysis of the article VoG: Summarizing and Understanding Large Graphs

Daniel Pollithy (LMU Munich)

***Abstract*—This is an analysis of the paper "VoG - Summarizing and Understanding Large Graphs"[1] by Danai Koutra (Carnegie Mellon University), U Kang (KAIST), Jilles Vreeken (Max Planck Institute for Informatics and Saarland University) and Christos Faloutsos (Carnegie Mellon University). VoG stands for Vocabulary-based summarization of Graphs. It is a near-linear algorithm proposed to deliver the most important structures a big graph consists of. After reflecting the article this analysis contains another experiment with VoG which did not come out as expected and sets the algorithm in context with the related work.**

I. Introduction

As of 2017 huge amounts of data are produced. A prominent example is the social network facebook.com with 1.3 billion users per day[2]. In 2014 they already stored more than 300 petabytes in their data warehouse[3]. There are plenty use cases for algorithms that try to discover semantically useful structures in graph data. For example determining fake accounts on facebook.com or important influencers on instagram.com. The authors of VoG saw the need of a flexible algorithm with near-linear time complexity.

They state that there are already many algorithms for community detection but their lack of labeling the found structures and ordering them by relevancy verifies the need. The proposed algorithm VoG focuses on a vocabulary of structures to describe graphs. Not only cliques and stars shall be possible but also chains, bi-partite cores and near-cliques. VoG was designed to be flexible so future structures that are semantically important can also be added. The output of the algorithm is an ordered list of important structures that can be interpreted by users with domain specific knowledge.

An informal description of the problem:

- Given a graph
- Find a set of (possibly overlapping) subgraphs to best describe the graph with a scalable algorithm

II. Outline

First off, the problem of graph summarization will be layed out. After that an introduction to MDL and MDL for graph summarization follows. The requirements for the algorithm preceed the actual step-by-step-guide. Then we focus on the complexity of VoG, the experiments made in the paper and a new toy experiment. In the final step we conclude the main ideas of the article.

III. Graph Summarization

Usually graphs are stored in adjacency matrices. There are approaches to compress them, like cross associations. Graph summarization is a different approach: It tries to represent a graph by its structures. The efficiency is still measured by its degree of compression according to Occam's razor: The less steps an explanation needs the likelier it is. An example: Johanna wants to send a graph to Alfonse through a bottle necked wire. Instead of sending the adjacency matrix she identifies the most important structures of the graph and sends them to Alfonse. Except for some error he will be able to reconstruct the graph.

The MDL (minimum description length) principle [4] can be used to formalize this 'efficiency' of compression. In general there is a model M and a function L which calculates the encoding cost of its input. So $L(M) + L(D|M)$ describes the cost of the model M and the Data D using the Model M. Using MDL we can compare lossless encoding methods.

For graph summarization the models are subgraph structures that are common in graphs. An example for such a structure could be a star, a clique...

As a result we obtain a minimization problem: Which model reduces the encoding cost of the subgraph the most?

IV. Requirements for the algorithm

A. General requirements

We are working on undirected graphs $G(V, E)$ of $n = |V|$ nodes, and $m = |E|$ edges, without self-loops. The theory could be generalized to graphs without these restrictions.

B. Graph vocabulary

To compose the graph summary, we make use of a set of graph structure types Ω called *vocabulary*. The vocabulary could be extended. The authors chose to work with the six most common structures in real-world graphs because they are well-known and understood by the graph mining community: *full* and *near* cliques (fc, nc), full and near bi-partite cores (fb, nb), stars (st), and chains (ch). Compactly, we have $\Omega = \{fc, nc, fb, nb, ch, st\}$.

C. Candidates

We consider ordered lists of graph structures as the Model M. The nodes may overlap but not the edges. Each structure $s \in M$ identifies a part of the adjacency matrix A.

Overlap between structures is allowed. That means that nodes can be part of more than a single structure. But edges are described on a first-come-first-serve basis: the first structure $s \in M$ to describe an edge (i, j) determines the value in A. There is no limit to the amount of overlapping edges. MDL just doesn't add a structure if it is too 'costly'.

$area(s, M, A)$ is defined as the edges of the adjacency matrix A $(i, j) \in A$ that structure s describes, where M and A might depend on the context.

C_x is the set of all possible structures of type $x \in \Omega$, and C the union of all of those sets, $C = \cup_x C_x$. C_{st} is the set of all possible stars. The model family $\mathbf{M}$ consists of all possible permutations of all possible subsets of C – recall that the models M are **ordered** lists of graph structures. We are looking for the $M \in \mathbf{M}$ that best balances the complexity of encoding both A and M using MDL.

D. Encoding the Model

The encoded length for a single Model $M \in \mathbf{M}$, L is defined as follows:

$$\begin{aligned} L(M) &= L_N(|M|+1) + \log \binom{|M|+\Omega|-1}{|\Omega|-1} \\ &\quad + \sum_{s \in M} \big(-\log \Pr(x(s) \mid M) + L(s) \big). \end{aligned}$$

1) $L_N(|M|+1)$ is the total number of structures in M using L_N (the MDL optimal encoding for integers ≥ 1)
2) $\log \binom{|M|+\Omega|-1}{|\Omega|-1}$: By an index over a weak number composition, we optimally encode the number of structures of each type $x \in \Omega$ in model M.
3) for each structure $s \in M$, we encode its type $x(s)$ with an optimal prefix code, and finally its structure.

To compute the encoded length of a model, we need to define $L(s)$ per structure type and also a heuristic how to find them in a subgraph:

1) Cliques: We call a set of nodes that is fully-connected a *full clique*. For a *full clique*, first the number of nodes is encoded and after that the list of ids:

$$L(fc) = L_{\mathbb{N}}(|fc|) + \log \binom{n}{|fc|}.$$

The number of ids is estimated in one go by an index over all possible selections of $|fc|$ nodes out of n total.

There could be one edge missing to interpret the structure as a full-clique. That is no problem. It can still be a full-clique as long as MDL selects it like this. The errors are stored in the error matrix E.

Labeling a subgraph as a clique is easy. The algorithm just connects all nodes within and calculates the error for missing edges.

2) Near-cliques: Related to the full-cliques are the (less dense) *near*-cliques. Their encoding is like this:

$$\begin{aligned} L(nc) &= L_{\mathbb{N}}(|nc|) + \log \binom{n}{|nc|} \\ &\quad + \log(|area(nc)|) + ||nc||l_1 + ||nc||' l_0. \end{aligned}$$

The transmition of the number, optimal prefix and ids is as above. We write $||nc||$ and $||nc||'$ for resp. the number of present and missing edges in $area(nc)$. Then, $l_1 = -\log((||nc||/(||nc|| + ||nc||'))$, and analogue for l_0, are the lengths of the optimal prefix codes for resp. present and missing edges. The intuition is that the more dense/sparse a near-clique is, the cheaper encoding it becomes. Note that this encoding is exact; no edges are added to E.

Labeling near-cliques are 'detected' same as full-cliques. Only the errors are stored individually.

3) Bipartite Cores: The definition of a Bipartite cores is non-empty, non-intersecting sets of nodes, A and B, which only have edges *between* the sets A and B, and not *within*. In other words, different from the off-diagonal rectangles, upon reading the description of a bipartite core, we can fill out $(i, j) \in A \times A$, and $(k, l) \in B \times B$ of M with 0s. How to fill out the area $A \times B$ depends on whether we read a full bipartite core (fb) or near bipartite core (nb).

The encoding of a fb (full bi-partite) is

$$L(fb) = L_{\mathbb{N}}(|A|) + L_N(|B|) + \log \binom{n}{|A|, |B|},$$

where first the size of A and B are encoded, and then the node ids.

Determining the bi-partite core in the subgraph is not trivial. The problem of finding the maximum bi-partite graph is known as max-cut problem, which is NP-hard [5]. The authors suggest to solve this by a semi-supervised classification: The prior knowledge is:

1) The highest-degree node belongs to A and its neighbors to B.
2) Heterophily: Connected nodes belong to different classes.

To propagate the classes, Fast Belief Propagation (FaBP) is employed [6].

4) Near bi-partite cores: For a near bi-partite core nb we have

$$\begin{aligned} L(nb) &= L_{\mathbb{N}}(|A|) + L_N(|B|) + \log \binom{n}{|A|, |B|} \\ &\quad + \log(|area(nb)|) + ||nb||l_1 + ||nb||' l_0. \end{aligned}$$

The determination is the same as for full-bi-partite core.

5) Encoding stars: A star can be seen as a specialization of the bipartite core. But it just has a single node in A which is called hub. That hub is connected to at least two nodes of the set B. Utter nodes are called spokes.

For $L(st)$ of a given star st we have

$$L(st) = L_{\mathbb{N}}(|st|-1) + \log n + \log \binom{n-1}{|st|-1},$$

where $|st|-1$ represesents the number of spokes. We identify the hub out of n nodes, and the spokes from the remainder.

Labeling a subgraph as a star is easy. The highest-degree node is the hub and the rest are spokes.

6) Encoding chains: A list of nodes where every node has only one edge to the following and one to the precedding node. In the adjacency matrix this looks like a super-diagonal (directly above the diagonal). As such, for the encoded length $L(ch)$ for a chain ch we have

$$L(ch) = L_{\mathbb{N}}(|ch|-1) + \sum_{i=0}^{|ch|} \log(n-i),$$

where the number of nodes is encoded first, and then the ids of the chain in the correct order. Note $\sum_{i=0}^{|ch|} \log(n-i) \leq |ch| \log n$.

Labeling a subgraph as a chain can be reduced to finding the longest path in a graph. This problem is also NP-hard [7] therefore we need a heuristic for it, which works as follows:

Try to find a long chain that goes through as many nodes as possible by employing heuristics. First pick a node at random and find its furthest node by executing Breadth First Search (BFS). This node is the temporary start of the "best" chain that describes the given subgraph. Then execute BFS again and find the furthest node to the start node; this is the temporary end of the chain. In order to further extend the chain found, apply one more heuristic; delete from the subgraph all the nodes that are part of the chain except for the endpoints. Given the induced subgraph, try to locally extend the endpoints by executing BFS and locating its furthest nodes. If new nodes are found in this step, concatenate them to the appropriate place (start or end) of the previously found chain, in order to get the extended chain.

E. Error matrix

It is likely that perfect structure type matches are rare. Therefore every structure candidate also carries an error cost with it.

In order to properly compare between models, MDL requires an encoding to be lossless. So the error matrix E's job is to make VoG's compression lossless.

E can be calculated by applying the exclusive OR between $\mathbf{M}$ and $\mathbf{A}$. For example: $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$.

Knowing M and E, the full adjacency matrix A can be reconstructed without loss.

We have to extend the cost function of the main score with the encoding cost of the error:

$$L(G, M) = L(M) + L(\mathbf{E}),$$

where $L(M)$ and $L(\mathbf{E})$ are the numbers of bits that describe the structures, and the error matrix E respectively.

The authors describe the formal definition of the problem:

Given a graph G with adjacency matrix A, and the graph structure vocabulary Ω, by the MDL principle we are after the smallest model M for which the total encoded length

$$L(G, M) = L(M) + L(E)$$

is minimal, where $E = \mathbf{M} \oplus \mathbf{A}$ is the error matrix, and M is an approximation of A deduced by M.

Next, we formalize the encoding of the model and the error matrix.

F. Encoding the Error

The encoding of the error is split up into two parts:

1) The area which is covered by the Model $\mathbf{M}$: E^+
2) The area which is not covered by the Model $\mathbf{M}$: E^-

They are encoded seperately because it is likely that they have different error distributions. Near bi-partite and near full-clique already have an error handling so they are ignored. E^+ only needs to be encoded for the areas containing full cliques, full bipartite cores, stars, and chains.

The encoding of E^+, and E^- is according to the one of near-cliques:

$$\begin{aligned} L(E^+) &= \log(|E^+|) + ||E^+||l_1 + ||E^+||'l_0 \\ L(E^-) &= \log(|E^-|) + ||E^-||l_1 + ||E^-||'l_0 \end{aligned}$$

The search space M consists of all possible permutations of the collection C of all possible structures over the vocabulary Ω. Miettinen and Vreeken showed that only finding the MDL optimal model of full-cliques is already NP-hard in a directed graph. [8] In order to keep the algorithm performing on big graphs the authors prefer to use heuristics for to 'solve' the minimization problem.

V. Proposed algorithm: VoG

The code for the algorithm is available for research purposes at www.cs.cmu.edu/~dkoutra/SRC/VoG.tar .

There are three steps:

1) Subgraph generation: Generate candidate subgraphs using decomposition methods (specifically SlashBurn [9])
2) Subgraph labeling: Characterize each candidate with a structure of the vocabulary (using MDL to decide which structure matches the best)
3) Summary assembly: Pick the best matching candidates using heuristics (Plain, Top-10, Top-100, Greedy'n'Forget)

And then return a graph summary M and its encoding cost. In the proposed framework, the second step of the algorithm can be explained with these steps:

1) Test whether the subgraph is an exact clique, star, chain, or bipartite core. If yes, include this structure in the candidate set C.
2) If not, reorder the nodes such that they are best represented as a star, chain, and bipartite core.
3) Compute the encoded cost for each representation (using the encoding scheme presented in the previous section), and include in the candidate set C the structure of the subgraph with the minimum cost.

Given a set of candidate structures, C, how can we efficiently induce the model M that is the best graph summary?

The last step of our summarization algorithm is the induction of model M through selection from the pool of candidate structures, C. The exact selection algorithm, which considers all the possible ordered combinations of the candidate structures and chooses the one that minimizes cost, is combinatorial, and cannot be applied to any non-trivial candidate set. Heuristics that will give a fast, approximate solution to the description problem are needed. To reduce the search space of all possible permutations, we attach to each candidate structure a quality measure, and consider them in order of decreasing quality. The measure that we use is the encoding benefit of the subgraph, i.e., the number of bits that are gained by encoding the subgraph as structure x instead of noise.

The baseline approach is to simply skip the structure selection process, and gives as graph summary all the candidate structures, i.e., $M = C$.

top-k: Selects the top k candidate structures, which are sorted in decreasing quality.

Greedy'n'Forget: Considers each structure in C sequentially, and includes it in M: if the graph's encoding cost does not increase, then it keeps the structure in M; otherwise it removes it.

VI. Experiments

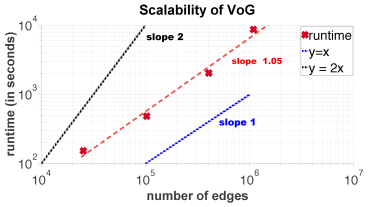

Fig. 1. VoG is near-linear on the number of edges

A. Time complexity

The authors designed the steps of VoG to be near-linear so the algorithm as such is also near linear Fig. 1.

B. Quantitative analysis

The authors conducted from the sheer amount of structures they found in the graphs and the fact that VoG works to some extent as a compression for the graph data, that the real world graphs do contain structures which can be mined with the given vocabulary.

C. Qualitative analysis

1) Graph summaries: The authors experimented on seven real graph data sets. They found out that the most frequent structures found by VoG in the graph are stars and near-bipartite cores. In the flickr and WWW-Barbasi graph they found a significant number of full cliques.

Furthermore they compared the summarization heuristics and came to the result that Greedy'n'Forget is good at dropping uninteresting structures.

2) Graph understanding:

1) The Wikipedia controversy graph: The most important results were stars and bi-partite cores which indeed were editors, heavy users and so called 'edit wars'
2) The enron email communication graph: The stars corresponded to the important figures in the company and the bi-partite core was about an extramarital affair.

VII. Additional experiment

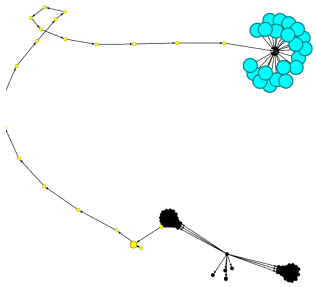

Fig. 2. The new synthetic data contains of two cliques, two stars and one chain

In order to test the implementation with a custom graph I made a small toy experiment with the code found on Github https://github.com/GemsLab/VoG_Graph_Summarization.

A. Synthetic data

The sample graph that came with the code consisted of 50 nodes. They were extended by a chain of 20 nodes and a star of 20 nodes at the end. Fig. 2

B. Unexpected outcome

Instead of delivering the obvious structures, the top-1 structure was a near bi-partite core followed by full-cliques.

C. Explanation and rating of the error

Disclaimer: Of course it is possible that a mistake was made producing the synthetic data or using the algorithm. It is also possible that a chain containing 20 nodes is just to long for real life graphs. But one can definitely deduct from this example that the employed heuristics do have down sides. If a structure seems to be 'good enough' the better ones are not found. An explanation why a near bi-partite core was selected here is that the center of the appended star is interpreted as the top node for the bi-partite core A. Because of the heterogeneity assumption the uneven nodes of the chain are added to one core and the even ones to another.

VIII. Related work

A. gSpan[10]: graph-based substructure pattern mining

The gSpan algorithm outputs frequent substructures in graphs and was initially created for use cases in chemical domains. It does not generate candidate structures and therefore does not interfere with the reason of existence of VoG.

B. Subdue

This algorithm was proposed in 2005 [11]. Subdue not only looks for frequent patterns but also for those compressing the graph efficiently. The main difference between Subdue and VoG is VoG's focus on a graph vocabulary. Which makes VoG more of a classifier than Subdue which can detect frequent substructures independent of given structures.

C. Unique features

The authors mention that there are already good graph partitioning and community detection algorithms that are specialized on some structures. But, according to them, what sets VoG apart are the following points:

1) Soft clustering
2) Scalability
3) A large vocabulary of graph primitives
4) Free of parameters

IX. Conclusion

The authors made the following contributions to the topic of big graph summarization and understanding:

- Information theoretic problem formulation
- Effective and Scalable Algorithm
- Experiments on real graphs

References

[1] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos, "Vog: Summarizing and understanding large graphs," in *Proceedings of the 2014 SIAM international conference on data mining*. SIAM, 2014, pp. 91–99.

[2] I. Facebook, "Results Conference Call," https://s21.q4cdn.com/399680738/files/doc_financials/2017/Q1-'17-Earnings-transcript.pdf, 2017, [Online; accessed 25-July-2017].

[3] ——, "Scaling the Facebook data warehouse to 300 PB," https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/, 2014, [Online; accessed 25-July-2017].

[4] A. Barron, J. Rissanen, and B. Yu, "The minimum description length principle in coding and modeling," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2743–2760, 1998.

[5] S. Poljak and Z. Tuza, "Maximum cuts and large bipartite subgraphs," *DIMACS Series*, vol. 20, pp. 181–244, 1995.

[6] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," *Exploring artificial intelligence in the new millennium*, vol. 8, pp. 236–239, 2003.

[7] D. Karger, R. Motwani, and G. Ramkumar, "On approximating the longest path in a graph," *Algorithmica*, vol. 18, no. 1, pp. 82–98, 1997.

[8] P. Miettinen and J. Vreeken, "Model order selection for boolean matrix factorization," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 51–59.

[9] Y. Lim, U. Kang, and C. Faloutsos, "Slashburn: Graph compression and mining beyond caveman communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3077–3089, 2014.

[10] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. IEEE, 2002, pp. 721–724.

[11] N. S. Ketkar, L. B. Holder, and D. J. Cook, "Subdue: Compression-based frequent pattern discovery in graph data," in *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*. ACM, 2005, pp. 71–76.