Oliver, Xander, Holden, and Jacob
COSC 254
5/19/2021

Summarizing Large Graphs with VoG

Introduction:

Our project focuses on the task of succinctly summarizing massive graphs. When presented with a massive graph of thousands or even millions of nodes and edges, what is the best way to describe and summarize the various patterns and subpatterns of the graph? In the paper "Summarizing and Understanding Large Graphs"[1] Koutra et al. propose VoG (Vocabulary-based summarization of Graphs) to help accurately and efficiently describe a given graph in terms of local graph structures. To do so, VoG approximates the MDL-optimal summary of a given graph in terms of the local graph structures. The local graph structures can be generated from any graph decomposition algorithm; we chose to implement SlashBurn[2].

Consider, for example, a large graph containing all of the edits made on a popular Wikipedia article where nodes represent individual editors and a shared edge represents edits to the same part of the article. A basic visualization of the resulting graph gives us almost no insight into the graph itself, but a summarization of the graph (using VoG) can reveal interesting information about the most important substructures of the graph. For example, we might find a list of the most important "stars" (one of six prevalent graph substructure types that the research paper outlines) which might indicate either Wikipedia super-users or admins who edited many different parts of the article. We might also get (as an output from VoG) a list of the most important "bipartite cores" which could indicate two groups of editors constantly overwriting one another's changes. This sort of pattern discovery from the summarization of large graphs is extremely useful not only in the context of Wikipedia but across large graphs of all types.

[1] Koutra et al., "Summarizing and Understanding Large Graphs."
[2] Lim, Kang, and Faloutsos, "SlashBurn."

Algorithm Description:

The primary goal of this project is to encode Vocabulary-based summarization of Graphs to describe large graphs. VoG relies on the Minimum Description Length for compressing the graphs. MDL compresses a dataset into a lossless subset that can completely represent the entire original dataset. VoG builds off of the MDL concept and applies it to graphs, using subsets of possibly overlapping subgraphs to succinctly describe a larger graph. MDL minimizes the length of the description of a model plus the length (in bits) of the description of that model when it is encoded with the data. VoG first uses a graph decomposition algorithm to split the original graph into subgraphs. The subgraphs generated from SlashBurn are then classified as one of six graph types from the following vocabulary: full clique, near clique, full bi-partite core, near bi-partite core, star, and chain. Each subgraph from SlashBurn is first tested against each vocabulary structure type above. If a perfect match for any of the above six structures is not found, the subgraph is then encoded as each substructure and is labeled as that which results in the lowest encoding cost according to the Minimum Description Length (MDL).

Once we have represented each subgraph as one of the vocabulary structures, MDL is used to associate the candidate structure with its encoding cost and is added to a candidate set **C**. Additionally, we compute the worst-case encoding cost of the subgraph (i.e. the cost we would get if we didn't encode it as its labelled subgraph type). The difference between this worst-case cost and the MDL encoding cost is then labelled as encoding benefit. The encoding benefit is then used to denote the quality of a given subgraph structure (higher quality = larger benefit).

Now, given a set of candidate structures **C** and their associated qualities (benefits), the heuristics "Plain", "Top K", and "Greedy 'N Forget" are used to select a non-redundant subset of candidate structures to represent the graph model **M**. The model of the heuristic with the lowest description cost is selected. Plain is the baseline implementation, which chooses the entire set of candidates as the summarization. Top-K chooses the top *k* candidate subgraphs, after sorting by decreasing quality. Finally, Greedy'N Forget chooses candidate subgraphs to add to the model based on whether or not the subgraph

increases the cost of the model. If a subgraph does not increase the cost of the model, it is added to the model.

Methods:

In our algorithm, we had four main components that we needed to code to get the entirety of the program to work. In the program, we had to encode SlashBurn, MDL, the error for MDL, and the various heuristics.

Slashburn is intended to help decompose the overall graph into a number of subgraphs. SlashBurn is given an adjacency matrix and returns a list of candidate subgraphs. The benefits of SlashBurn are that it is scalable for graphs of any size and can find candidate substructures that are not always clearly defined or exactly one type. We encoded a greedy and regular version. If the greedy algorithm is selected, the algorithm will be slightly slower but also more accurate.

For our Minimum Description Length algorithm we attempt to define the best summary of a graph by looking at the set of subgraphs that most efficiently and accurately describes the various candidate subgraphs. MDL uses the database in order to compress a dataset into a lossless subset that can completely represent the entire original dataset. To encode the minimum description length, we had to encode a different way of calculating the length for each possible graph in our vocabulary. The general formula follows $L(G, M) = L(M) + L(E)$ where M is the model and E is the error. To generate $L(M)$ we must use the different formulas provided in the paper. For example, the length of a full clique can be given by: $L(fc) = L_N(|fc|) + log(\binom{n}{|fc|})$, where $|fc|$ is the number of edges in the clique, and n is the number of nodes. The equation for a bipartite core can be given with $L(fb) = L_N(|A|) + L_N(|B|) + log(\binom{n}{|A|, |B|})$, where n is defined as before, and $|A|$ and $|B|$ are the sizes of the two component sets of nodes. Each of the equations follows the same general

pattern with small differences for each type in the vocabulary, as well as showing differences between graphs that are “near” versions graphs that approximate an exact type of graph, such as a near-clique versus a full clique.

To calculate the error cost, we must calculate E^+ and E^- which represent the area that M does and does not model respectively. Both values follow the same equation in which $L(E^+) = log(|E^+|) + ||E^+||l_1 + ||E^+||l_0$. $L(E^-) = log(|E^-|) + ||E^-||l_1 + ||E^-||l_0$. These formulas closely mirror that of near-cliques. This method is called the “prefix” method which is preferred over a binomial approach as the “prefix codes allow us to easily and efficiently calculate accurate local gain estimates in our algorithm.”

For our heuristics, encoded three different methods for selecting the best graph summary. The plain method just looks at all the graph summaries of candidate structures with $M = C$. The Top-K heuristic selects the best *k* structures. Finally, the “Greedy ’N Forget” heuristic considers each summary in descending quality order, and if the encoded cost does not increase, it keeps the summary and otherwise ignores it until all candidates have been considered.

Results:

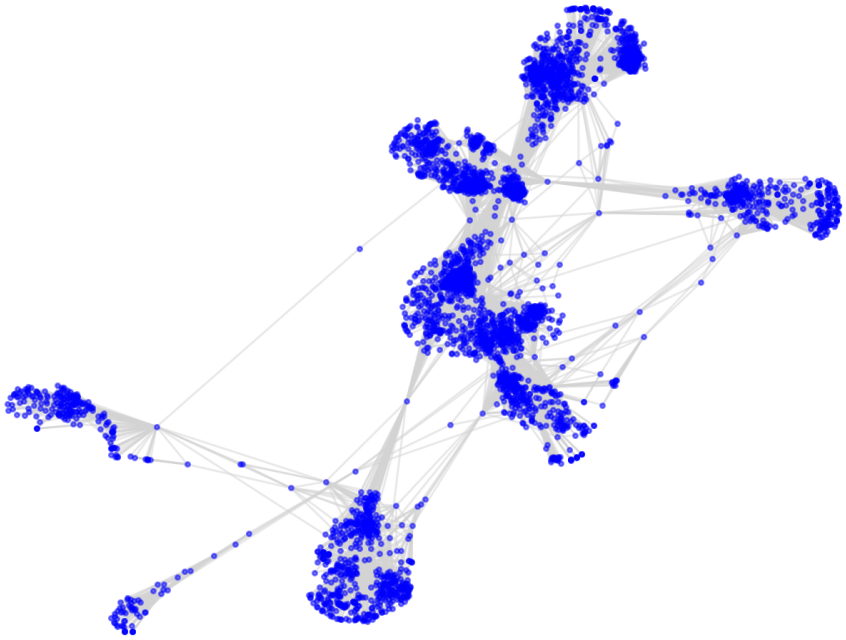

Figure 1*: The Facebook Social Circles Graph[3] before compression. This graph has 4039 nodes and 88234 edges. It consists of friend lists from Facebook where each node is a user and each edge between nodes represents a "friendship". Note that this graph is the same as the model selected from the Plain heuristic because we run SlashBurn optimally (it generates subgraphs that perfectly represent the original graph).*

[3] "SNAP: Network Datasets: Social Circles."

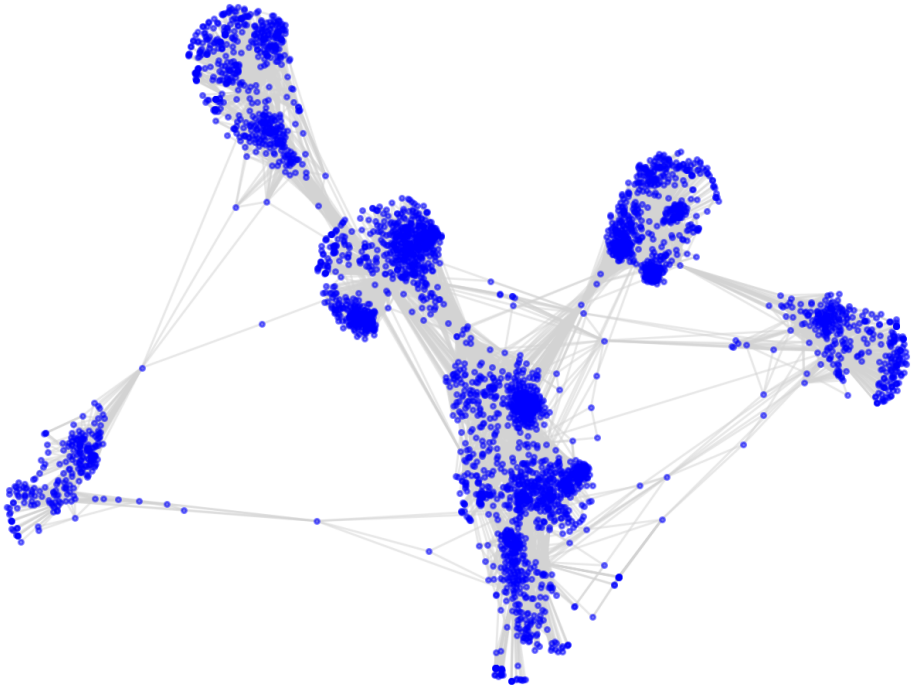

***Figure 2**: The Facebook Social Circles Graph after running VoG and selecting a model using the Top K heuristic with k=10.*

***Figure 3**: The Facebook Social Circles Graph after running VoG and selecting a model using the Greedy 'N Forget heuristic (note that this is the same as the original graph).*

Figures 1-3 show the graphs resulting from the compression by the three heuristics. The Plain heuristic just uses all the candidate structures (2207 in the case of this graph) and since SlashBurn perfectly decomposes the original graph into subgraphs, the resulting graphs are both the exact same. The Top-K heuristic (in this case the top 10 structures), created a graph that is significantly different from the original due to the fact that there are significantly fewer structures being represented. Finally the Greedy ‘N Forget heuristic only removed 100 structures from the original 2207, and since the subgraphs can be overlapping the graph looks identical to the original graph. The fact that this heuristic only removed 100 subgraphs seems to indicate that there may be an issue in the way candidate structures are added to the model. We believe this is because we do not calculate total model error correctly.

Graph Label	**Number of Structures**
Star (st)	1465
Full Clique (fc)	739
Near Clique (nc)	2
Chain (ch)	1
Bipartite Core (bc)	0

Figure 4: *Breakdown of labelled graph structures from the Facebook Social Circles Graph.*

The way we categorized structures was if they did not perfectly fit any graph type we chose the best approximation. In this case, cliques and stars were the best approximations for graphs that did not fit an exact type which intuitively makes sense, because Facebook friends will all be connected to each other. Additionally one person could be connected to many people which would closely represent a star structure. As a result, star and full clique were the most commonly found structures.

Heuristic	**Total Model Error (bits)**

Plain	702,760
Top 10	**702,743**
Greedy ‘N Forget	702,832

Figure 5: *Breakdown of total model error (in bits) from the three heuristics run on the Facebook Social Circles Graph (lower is better).*

Looking at Figure 5 above, the Top K heuristic results in the lowest total model error, which implies that for the Facebook graph, it is the best heuristic. However, the paper indicates that the Greedy ‘N Forget typically chooses the best model due to the simplicity of the other two heuristics. Additionally, Top-K and Plain are both subcases of the Greedy ‘N Forget heuristic, so we would expect the chosen model (in all cases) to be the one from Greedy ‘N Forget. Because this is not the case, we believe that we must be making some mistake in our total model error calculation.

Key Decisions

- After converting the initial file into an adjacency matrix, SlashBurn outputs the resulting subgraphs as a list of lists.
- These subgraphs were then evaluated to see if they fit perfectly into one of 5 subgraph categories, star (st), full clique (fc), near clique (nc), chain (ch), and full bipartite core (bc). We did not check to see if a subgraph was a near bipartite core since the algorithm to do so is NP-hard. We also chose not to check if a subgraph was a near chain, since the algorithm for this check is also NP-hard. If it did fit exactly into one of these categories, the MDL cost can be calculated and subtracted from the noise cost to get the encoding benefit (quality metric).
- If a subgraph did not fit any of these types, we had to choose the best approximation. However, the approximation for identifying a subgraph that is almost a bipartite core, or almost a chain requires machine learning to do efficiently so we did not try and code them. Instead these

approximations could fall into one of three categories, almost a clique, almost a star, or almost a near clique. At this point the local error cost can be calculated and added to the MDL cost.

- Finally we outputted the parameters of the graph type (st, ch,...) the list of nodes for the subgraph, the noise cost, and the encoding cost and stored them in pairs within dictionaries. Each dictionary had subgraphs as its value. This was chosen in order to match up subgraphs with their graph type and their encoding benefit.
- With this information we moved on to use the three heuristics: PLAIN, Top K, and Greedy ‘N Forget inorder to select the candidates that are the most informative about the graph. For the Greedy ‘N Forget heuristic a total model error also needs to be computed in order to determine if a subgraph should be included in the model.
- Overall, the three most common structures we used were, a list of lists to represent subgraphs, dictionaries to connect properties like label and cost to subgraphs, and sets were used in many functions due to its fast lookups and the properties that duplicates will not be added into sets.

Limitations and Future Work:

For this project, we were limited by a variety of factors. First, the paper that our project was based on gave mostly broad mathematical overviews of how to code different elements of the code, with few actual guidelines on the best implementation. This leads to many situations in which we were left to make our own decisions and guesses about the optimal implementation. As a result, our code is quite different from the author’s code, even if the overall goal is the same. Furthermore, the authors would sometimes mention some version of implementation or advice about coding practices, but not actually follow those same guidelines in their code. We were also limited by our own knowledge of complex graphs, as none of us have the same background or experience as the authors.

In terms of further work, our project was limited by the amount of time we had on hand. Our implementation is quite slow and leads to us needing to use smaller or medium-sized graphs, although it

should also work on larger graphs as well. This would also allow us to gather more general and in-depth results that could be further applied to all graphs of any size. Another area that would have been interesting to explore would be using different metrics for the quality of each subgraph. In our implementation, we describe the quality of a subgraph as the difference between the noise cost and the MDL encoding cost. With more time, we would want to explore how using just the MDL encoding cost as the quality of a subgraph changes the models outputted from the three heuristics.

Bibliography:

1. Koutra, Danai, U Kang, Jilles Vreeken, and Christos Faloutsos. “Summarizing and Understanding Large Graphs.” *Statistical Analysis and Data Mining: The ASA Data Science Journal* 8, no. 3 (June 2015): 183–202. https://doi.org/10.1002/sam.11267.

2. Lim, Yongsub, U. Kang, and Christos Faloutsos. “SlashBurn: Graph Compression and Mining beyond Caveman Communities.” Knowledge and Data Engineering, IEEE Transactions On 26 (December 1, 2014): 3077–89. https://doi.org/10.1109/TKDE.2014.2320716.

3. “SNAP: Network Datasets: Social Circles.” Accessed May 19, 2021. https://snap.stanford.edu/data/egonets-Facebook.html.