General Social Pattern Discovery

Bhopesh Bassi
University of Illinois - Urbana Champaign
bbassi2@illinois.edu

Tianhang Sun
University of Illinois - Urbana Champaign
ts7@illinois.edu

Dustyn Tubbs
University of Illinois - Urbana Champaign
dtubbs2@illinois.edu

Jingnan Yang
University of Illinois - Urbana Champaign
jingnan2@illinois.edu

Jia Wang
University of Illinois - Urbana Champaign
jiawang4@illinois.edu

ABSTRACT

Community detection algorithms are tools that allow researchers to identify clusters of nodes that best characterize a network. In community detection, algorithms typically distinguish communities by analyzing the interconnectedness between nodes, regardless of substructures within these communities and, typically, attributes on these nodes. In this paper, we develop a linear-time process called General Social Pattern (GSP) discovery. By allowing for users to define heterogeneous substructures within communities, or "typed-motifs", GSP allows users to have fine-grain control most atomic piece of a community.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: [Graph and tree search strategies]; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Experimentation, Design, Algorithms

Keywords

Social Networks, Graph Mining, Community Discovery

1. MOTIVATION

Graphs serve as important data structures in many applications, including social network, web, and bioinformatics applications. An increasing number of applications work with web-scale graphs. Data sets originating from many different real world domains can be represented in the form of interaction networks in a very natural, concise and meaningful fashion. Analysis of such networks can result in the discovery of important patterns and potentially shed light on important properties governing the growth of such networks.

With the problem of community detection, the goal is to identify subgraphs of the network that are best characterized by their physical structure (e.g. the number of edges shared). This definition, however, does not take into consideration the type of network under examination nor additional properties on the nodes. Indeed, complex networks from different sources such as social networks, biological networks, communication networks, and electronic circuits typically display different statistically significant, recurrent subgraphs known as network motifs [2].

Motifs allow individuals to strictly define a graph structure which can be used to compare other graphs, similar to that of a regular expression. With this in mind, our goal then became how can we offer such level of control for the problem of community detection? To integrate these two concepts, we first work off of the notions presented in [4]. As a preliminary bit of knowledge, a Truss is just a simple structure that is that of a triangle formed by three nodes creating a cycle. A K-Truss is a network formed where the nodes in such a network share an edge that participates in a total of K truss'. Thus, we have our motivation:

Discover communities of structural and semantic similarity by way of typed network motifs. First, by allowing a user to describe the most atomic piece of their community by way of a motif, they can uncover more interesting communities than by way of just having simple, dense connections between nodes. Secondly, by identifying the queried motif inside of the network, we can exploit the work of K-Truss by building out a network of motifs which we can then perform simple clustering on. Lastly, by allowing a user to have attributes on each node in the queried motif, it allows the user to have far more precision in defining what a community is.

2. PROBLEM DEFINITION

We adopt the following definitions:

I	An input motif
G	An input graph from which to derive communities
A_n	The set of attribute values on the node n
E_X	The edges of the graph structure X
V_X	The vertices of the graph structure X

Our goal is to first find $M = \{m | E_m \subseteq E_G \wedge V_m \subseteq V_G \wedge m \simeq I\}$. That is, find M which is the set of all motifs in our

network that are isomorphic to the input motif. In addition to this structural isomorphisim, the isomorphic function $f : a \mapsto b$ needs to also verify that $\forall x \in A_a, x \in A_b$. That is, the attributes on each node in the input motif are also attributes found on all motifs found in G (wherein a represents a node in the original graph and b represents a node in the queried motif).

Utilizing this set of motfis, construct a network similar to [4] wherein we identify nodes in this network as an entire motif in M, and the edges weighted to be the number of overlapping edges that the two motifs share. Lets call such a network C.

Lastly, utilize a clustering algorithm to identify communities in a similar manner as before (*i.e.* define communities to be groups of nodes with dense edges). Given the set of nodes that were identified to be in dense clusters, all must do afterwards is just decompose the resulting set of motifs back into their original form in the network. Thus, our system then identifies clusters composed at the finest granularity of the definition of I.

3. SOLUTION

The overall framework of the approach is composed of three parts. In section 3.1, the subgraph matching algorithm is introduced and used for querying all occurrence of typed motifs in original network. Moreover, we introduce a novel extension of this subgraph matching algorithm, which supports only one attribute per node, to support multiple attributes. Section 3.2 includes how to construct a new motif network with the motif discovered and how to cluster the network into a constructed one.

3.1 Motif Discovery

Many existing approaches for graph query processing do not work for very large graphs. Typically, indexing on edges for graph queries ends up using many costly join operations and expressing only a subset of subgraph queries.

Our motif discovery algorithm is based on STwig approach in the paper [5]. The advantages and details of this approach is introduced as follows.

Provided the adjacency list, the algorithm will first construct two mappings between the node id and their attributes. That means given a node id, we can efficiently find all its attributes. Given an attribute, we can efficiently find all nodes that have this attribute. The approach in [5] only suppports one node with one attribute. In our approach, we extend the original algorithm so that one node can have multiple attributes.

To support multiple attributes, there is a match STwig phase introduced. STwig is a subcomponent of original graph motif that is a two level tree, by simplifying the subgraph matching to tree matching like this, it greatly increases the speed of the algorithm. In STwig phase, the algorithm will first find all nodes that have the root attribute. For example, if the root node of the two level tree is "UIUC", the algorithm will find only nodes with attribute "UIUC", and all others nodes remain untouched. After that, the algorithm will retrieve all their neighbors to check if they match the second level of the STwig. By checking the neighbors, each run can find all STwigs efficiently.

In our project, we still need to manually de-compose the motif into several sub-components, and feed the algorithm with STwigs. After retrieving all these STwigs, we can join all these STwigs to get the graph motifs. In the original algorithm, since each node can only have one label, the motif we get here is a mathcing motif. However, by extending label to attribute, this phase may result in some false positives. To counter this, we add another verification phase to check if each motif is a valid motif, and the resulting algorithm works correctly. During the project, we only used triangle, or star motif, where both of them are just a two level tree. In this case, we optimize the algorithm to avoid the expensive joining phase so that the motif retrieve always takes sub-second time. If the motif size is increased, the time complexity will increase dramatically. In specific, the worst time complexity of this addition is $O(n^2 log_2 n)$ with respect to the motif size. In general, the algorithm runs in linear time respect to the network size, and to the motif size.

3.2 Clustering

In order to perform community detection utilizing these motifs, we create the network C. To do this, we create a new network where each motif in the original network is collapsed into a node and edges between two such nodes are weighted as per the design of [4]. Once we have this network, we can use a clustering algorithm to find motifs that are closely linked and hence form a community. This network was created with two passes over all the motifs. In the first pass, we build a mapping from node to all the motifs it occurs in, and in second pass, we probe this mapping and keep building the new network.

Why need clustering? It is found that many networks display community structure-groups of vertices within which connections are dense but between which they are parser. Some high sensitive computer algorithms are computationally demanding. In our approach, we use the algorithm introduced in this paper[3]. This algorithm is based on the idea of modularity and has the worst time complexity of $O((m+n)n)$ or $O(n^2)$ on a sparse graph. This is a greedy agglomerative hierarchical clustering algorithm. The alogorithm efficiently clusters large number of nodes (this is one of the best scaling clustering algorithms) while producing a suggested number of clusters. This algorithm is implemented as a tool in python, called *AgglomCluster*.

Our overall solution can be best described by Algorithm 1.

4. EXPERIMENT

In this section, we report our experiment results that validate the effectiveness and efficiency of our approach. The clustering results is compared with the clustering results obtained from the original dataset.

Our experiements are based on the EgoNet-UIUC dataset [1], which has 29040 nodes and 58080 edges. The nodes in this dataset have attributes like employers, education and locations. As a baseline measurement, we run the algorithm in [3] on the network without an input motif to have a baseline measurement.

Data: The input motif $I = (V_I, E_I)$,the input graph $G = (V_G, E_G)$
Result: $G' = (V_{G'}, E_{G'}) \wedge \forall v \in V_{G'}, \exists vi \in V_I$ s.t. $A_{vi} \subseteq A_v \wedge v \cong vi$
begin
  $M \longleftarrow FindTypedMotifs(G, I)$
  $V \longleftarrow \emptyset$
  for $m \in M$ **do**
    if *m is structurally and semantically isomorphic to* I **then**
      $V \longleftarrow V \cup \{m\}$
    end
  end
  $Edges \longleftarrow \emptyset$
  $Nodes \longleftarrow \emptyset$
  for $v \in V$ **do**
    for $c \in V$ *where* $V_c \subset V_v \wedge c \neq v$ **do**
      if $(c, v) \in Edges$ **then**
        increment the weight of the edge (c, v, x)
      end
      else
        $Edges \longleftarrow Edges \cup \{(c, v, 1)\}$
      end
    end
  end
  Return $Cluster(\{Nodes, Edges\})$
end

Algorithm 1: General Social Pattern Discovery. The function `FindTypedMotifs` is our extension on [5]. The function `Cluster` is the algorithm described in [3].

To test clustering on motif network, firstly we manually generate several kinds of general motifs. In this section the triangle motif and the star motif are used as examples. Both of them are single level motifs (meaning, the nodes have only one attribute). The statistics of generating clusters based on these two kinds of motifs are shown in **Table 1**.

	T1	**T2**	**T3**	**N1**	**N2**
No Motif	N/A	N/A	2738	N/A	77
Triangle 1	0.208	0.505	10.733	1564	11
Triangle 2	0.260	0.473	10.456	1636	5
Star	0.456	4.813	38.832	1753	2

Table 1: Experimental results.
T1: Time to query graph for input motif (s).
T2: Time to construct motif network (s).
T3: Time to cluster network (s).
N1: Number of queried motif found.
N2: Number of cluster identified.

	Clusters	**Nodes**	**Edges**
Original	77	29040	58080
Triangle Motif	5	4820	20993
Star Motif	3	1753	3704

Table 2: Evaluation of cluster characteristics utilizing no motif to query (baseline, original row), the triangle motif (Figure 1), and the star motif (Figure 2).

Table 1 showcases the runtime performance of our system. The first row, No Motif, is included as a baseline measurement as to the performance of the clustering algorithm without utilizing an input motif. Rows two and three, (Triangle 1 and Triangle 2) showcase the performance of our system utilizing a triangle like structure as the queried motif (see Figure 1, which corresponds to Triangle 2.) Based off of these measuresments, we observe the following:

1. The running time of clustering on the motif network is much less than that without motifs. Also, the number of clusters reduces when motifs are introduced into the approach. This makes it more easier to find meaningful communities we really need. In addition, this we observe that by filtering the network for only our queried motifs, we reduce the problem space of the performance bottleneck that is the clustering algorithm.
2. With the complexity of the motifs increases, the running time of each stage increases consequently. In the meantime, the number of clusters reduces. Because the more complex the motif, the harder to find a community dense with this motif.

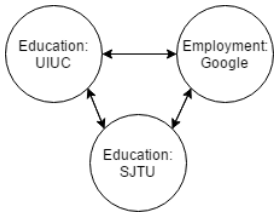

Figure 1: A simple triangle motif composed of three nodes, each having different attributes. Two nodes have attributes relating to their education selected (sjtu and uiuc alumni), whereas the other nodes has an attribute relating to their employment selected (google).

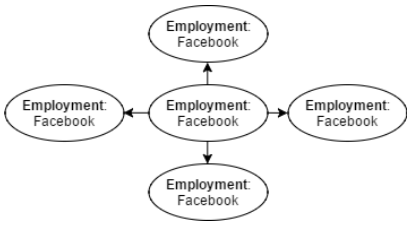

Figure 2: A more complex structured motif, but all nodes having an employment from Facebook. A query based off of this motif is looking for communities formed by individuals who all have worked at facebook and all have a hierarchical relationship with another individual (as an example, a manager).

The experiments listed above demonstrate the efficiency of our approach. In addition to efficiency, we also wanted to get a feel as to the semantic trends found in these communities. Figure 3 shows these results for the communities identified without using a motif to cluster. To contrast, the results of using two motifs (a triangle motif (Figure 1) and a star motif (Figure 2) are included.

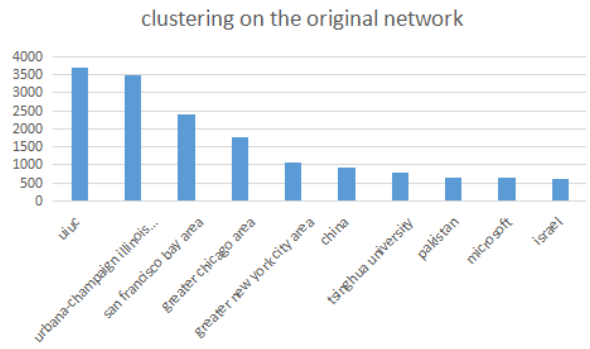

Figure 3: Top 10 most frequently occuring attributes when performing just simple clustering (without using a motif).

As we can see in Figure 3, attributes most commonly found in the communities derived without using a motif are randomly distributed between locations, companies, schools, and countries. Indeed, these attributes are more representative of the entire network and can't easily be used to characterize the communities found.

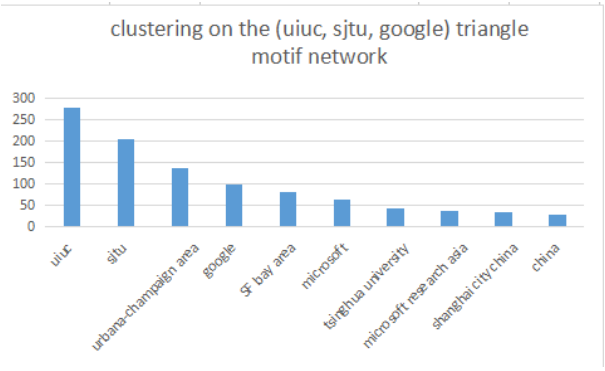

Figure 4: Top 10 most frequently occuring attributes when utilizing the triangle motif shown in Figure 1.

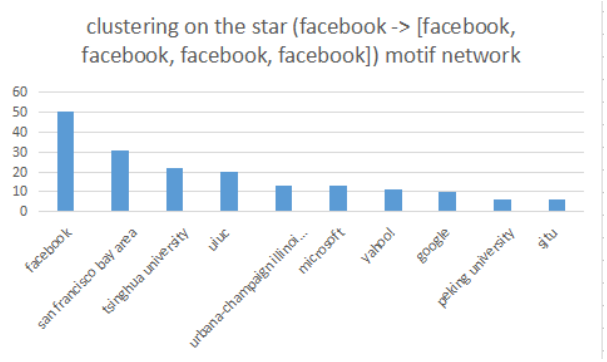

Figure 5: Top 10 most frequently occuring attributes when utilizing the star motif shown in Figure 2.

Contrast this with the most common attributes found utilizing the triangle motif (Figure 4), or utilizing the star motif (Figure 5), we can truly grasp how large of an impact the queried motif has. That is, by establishing some common attribute values using a motif, the other more frequently occuring values actually start to have some meaning.

As an example, take results shown utilizing the triangle motif. Just by looking at the aggregate statistics of our communities, we find the interesting trend that individuals who are participant in a triangle between an education at UIUC, an education at SJTU, and an employement at Google, typically have worked at Microsoft. This is a big improvement upon seemingly random attributes found just by classifying a community as a dense cluster of nodes.

5. LESSONS LEARNED

Overall, we feel as though the results that this work has taught us are qualitative and quantitative in manner. Qualitatively, we see the impact that utilizing a motif to define communities gives us:

1. The input motif used to define communities has such a huge impact in terms of the number of nodes found in the resultant community. If you try to cluster a network using a very particular motif (read large, odd attributes), you shouldn't be surprised if you don't find that many communities. Table 2 in our results showcases this very well.

2. By including types on the input motif to create communities, aggregate statistics such as the top-k most common attribute starts to become more interesting.

Quantitatively, this work has been a good demonstration as to the importance of minimizing your feature space when it comes to computational bottlenecks. Our adaptation to [5] only matters when the resultant graph fed to the clustering algorithm in [3] becomes smaller. We have such an incredible speed up because the huge bottleneck is the worst case $O(n^2)$ found in the clustering algorithm. By identifying motifs and creating the motif network in linear time with respect to the network size, we minimize the overhead caused by clustering.

6. FUTURE WORK

Having fine grain control over community detection is a powerful thing. However, there were some limitations of our techniques, as well as other bits of information available to use that we didn't end up using. Thus, in our minds, there are three major avenues for improvement: building an attribute hierarchy, making use of motif network weights, and performing on the fly community detection. In addition to these implementation improvements, another obvious improvement would be to use `C++` in order to make as much use of the hardware as possible.

6.1 Building an Attribute Hierarchy

Presently, our algorithms view both the input motif's attributes and the target network's attributes in a "flat" manner. That is, having the presence of an attribute is very strict, the target node needs to have the said attribute in verbatim or it will not be matched for the motif-network building phase. Instead of this, we propose that a future

avenue of improvement be to integrate an input network generated ontology to use for attribute matching.

The overall idea with this is that we want to provide more flexibility to a user for defining what a community could look like. As an example, say a user were to define a motif where all nodes have a college education. Clearly, the user is intending to identify communities of college-educated individuals of a certain structure. However, our current system doesn't support such a hierarchical notion, but instead only literal attribute values (*e.g.* having attended the University of Illinois - Urbana Champaign).

6.2 Taking advantage of Motif Network Weights

As we stated earlier, when we form the motif network, we create an edge between two nodes whose weight is the amount of overlap that the two motifs share of their edges. When we look to perform clustering, however, we don't end up making use of that information in order to determine what a community is. We hypothesize that taking advantage of this latent information would allow for a stronger definition for a community in our system.

6.3 Performing on the fly community detection

Our system works by performing its analysis offline. That is, in a set of discrete steps, cull the original graph of non motif nodes, generate the motif network, cluster the motif network, and then yield all nodes that are present in the clustered network. However, for a majority of systems, growth is organic and, thus, communities should be detected as they form. One way we believe this improvement could be relatively straight forward would be to store the computed motif network. As we look to add a node to the graph, check to see if it participates in the queried motif. If so, build on the motif-network and perform clustering once more.

7. REFERENCES

[1] R. Li and K. C. Chang. Egonet-uiuc: A dataset for ego network research. *CoRR*, abs/1309.4157, 2013.

[2] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[3] M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[4] J. Wang and J. Cheng. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment*, 5(9):812–823, 2012.

[5] H. W. B. S. Zhao Sun, Hongzhi Wang and J. Li. Efficient subgraph matching on billion node graphs. *Proceedings of the VLDB Endowment*, 5(9):788–799, 2012.