Semi-Supervised Community Detection Using Structure and Size

Arjun Bakshi, Srinivasan Parthasarathy, Kannan Srinivasan
Department of Computer Science and Engineering
The Ohio State University
Columbus, USA
{*bakshi.11, parthasarathy.2, srinivasan.115*}*@osu.edu*

***Abstract*—In recent years there have been a few semi-supervised community detection approaches that use community membership information, or node metadata to improve their performance. However, communities have always been thought of as clique-like structures, while the idea of finding and leveraging other patterns in communities is relatively unexplored. Online social networks provide a corpus of real communities in large graphs which can be used to understand dataset specific community patterns. In this paper, we design a way to represent communities concisely in an easy to compute feature space. We design an efficient community detection algorithm that uses size and structural information of communities from a training set to find communities in the rest of the graph. We show that our approach achieves 10% higher F1 scores on average compared to several other methods on large real-world graph datasets, even when the training set is small.**

***Index Terms*—community detection;semi-supervised;**

I. Introduction

Community detection can be informally described as a problem of finding subgraphs in a large graph where the connectivity of nodes within the subgraphs differs significantly from those outside them. Most approaches for community detection use metrics like modularity at their core, which are borrowed from social network analysis [1]. Such algorithms work by finding subgraphs that have high modularity. Some semi-supervised approaches incorporate sparse or partial information about communities into the detection process, while still trying to maximize modularity. However, a recent study by Fortunato et al. shows that *techniques that leverage modularity tend to perform poorly on real world graphs where communities are defined on latent variables or node metadata, rather than the cohesiveness of nodes in the graph [2].*

Online social networks allow us to collect node metadata and real communities in large graphs. Although some work has been done in incorporating metadata into community detection [3], the idea of leveraging a sample of known communities is relatively unexplored, despite many such datasets [4]. Furthermore, in some cases only the graph's structure is observable, and metadata is not. Therefore, in this paper we focus on cases where **metadata is not available but some communities in the graph are known.**

Given a graph and a small set of known communities in the graph, our goal is to extract more communities that have similar characteristics as the known communities. To do that, the following challenges must be addressed: *How can one characterize the information embedded in the sample of known communities?* Subsequently, *how does one incorporate these characteristics into a community detection algorithm?* We hypothesize that there is significant information embedded in the **size and structural composition** of communities that can improve community detection. Our analysis of graphs with known communities for 5 real world graph datasets supports these hypotheses, and is presented in Section II and III.

Motivated by these ideas, we present **Bespoke**, a "custom-fit" semi-supervised community detection algorithm. Bespoke models structural composition of known communities by computing fingerprints based on the structural features of the nodes and edges in them. It then extracts patterns in size and structural composition across the known communities, and uses them to search for similar structures (communities) over the complete graph.

II. Motivation

A. Patterns in Size

We use five graph datasets with known communities available on SNAP [4]. The details of the datasets used are presented in Table I. Figure 1 plots (a portion of) the size distributions of communities in the different datasets. While the general trend of smaller communities being more likely is clear, some datasets lean more towards smaller communities than others. For example, communities with 20 or more nodes form only $\sim$5% of all communities in DBLP, while for Facebook that number is $\sim$30%. These size variations motivate the idea of **extracting communities that follow the size distribution of the communities in a training set.** The intuition is that if communities tend to be in a certain range of sizes, then extracting communities that are much larger or smaller than those sizes will result in either low precision, or low recall with respect to real communities. Finally, although a training set may not model the entire community size distribution, (as can be the case with power-law distributions) **it can still provide a good estimate within some bounded interval.**

B. Patterns in Structural Composition

The idea of assigning roles/labels to nodes based on structural properties has been of recent interest [5]. Each node in

Dataset	Nodes	Edges	Comms
Amazon co-purchase	334,863	925,872	5,000
DBLP co-authorship	317,080	1,049,866	5,000
YouTube (YT)	1,134,890	2,987,624	5,000
Twitter (TW)	81,306	1,768,149	3,662
Facebook (FB)	4,039	88,234	191

TABLE I
DETAILS OF GRAPH DATASETS.

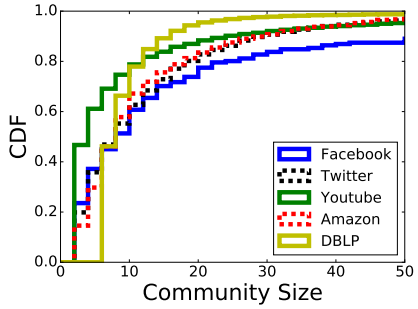


Fig. 1. Distribution of community sizes across datasets. DBLP generally has smaller communities compared to Facebook.

a graph fulfills a role in it, and the semantic meanings of the roles depend on the graph's domain. We build upon this idea and suggest that **a community can be characterized by the types and distribution of roles in it, and the connections between them.** Furthermore, if a clear pattern in the connections between roles is observed inside the training communities, then it can be leveraged to identify communities in the rest of the graph.

III. PRELIMINARIES

A. Modeling Structural Information

Consider a graph with labeled nodes, and a community in it. Our goal is to concisely represent information about the community structure, or its constituent nodes and edges.

Figure 2 shows an example of how we represent a community using labeled edges. An edge is labeled using the labels of the nodes that it connects. That is, an edge that connects nodes n_i and n_j with labels A and B respectively, is labeled as (A, B). First, we use the nodes in a training community to induce a subgraph. We then compute a **probability distribution of the labeled edges in the induced subgraph**. We refer to this distribution as a community's **feature vector** or its **features**.

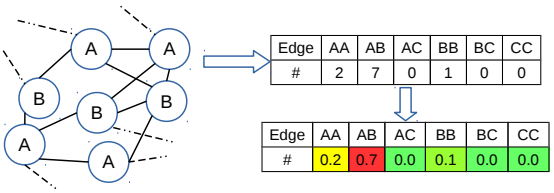


Fig. 2. Generating the feature vector of a community. The community's induced subgraph is drawn with solid lines, and edges to the rest of the graph with dashed lines. Although the graph has 3 possible node labels, the community contains nodes with only 2 of those. The probability distribution of the edge types in the community is used as a feature vector of the community.

The intuition behind creating a feature vector based on edges instead of nodes is the following: Consider a node set with n labeled nodes. The number of different possible graphs that one can construct on this node-set is vast ($O(2^{{}^nC_2})$). However, by putting constraints on how many edges can be induced between nodes with certain labels, that number can be greatly reduced. Therefore, **for a given labeled graph, an edge based representation can more closely (though not uniquely) represent the structure of the graph compared to a node based representation.**

B. Node Labeling

Although existing methods can be used for labeling nodes [5], we compute labels based on coarser local topological features in favor of lower time complexity. We first compute the Jaccard similarity of a node's neighborhood to that of each of its neighbors'. The similarity of two nodes is the number of neighbors they have in common divided by the total number of unique neighbors. This results in a distribution of Jaccard similarity scores for each node. A node with n neighbors will have n Jaccard similarity scores associated with it; one for each neighbor. This distribution captures the connectedness of a node with its neighbors. We compress this distribution into a feature vector for a node by using its $0^{th}, 25^{th}, 50^{th}, 75^{th}$ and 100^{th} percentile scores.

We choose this representation as it models the connectedness of nodes in an ego-net. For example, a node in a clique will have a very high value in the the 100^{th} percentile column. This is because all of its neighbors have the same neighborhood, leading to a distribution of very high ($\sim$1.0) Jaccard similarity scores. Similarly, the 100^{th} percentile value for a node that is mostly disconnected from its second order neighbors will be low.

The nodes are then clustered and labeled using K-Means clustering (K=N_l, number of labels). Based on the case studies in [5], the typical number of roles in most networks ranges between 3 and 5. For this reason, all of our analysis and experimental results use $N_l = 4$.

C. Patterns in Communities

We now compare the patterns seen in community subgraphs and random subgraphs using the aforementioned feature space. The number of node labels is set to 4 resulting in a feature vector of length 10. To generate a random subgraph, we pick a node at random and expand in a breadth-first fashion till T unique nodes have been visited. T is chosen **uniformly at random** from the $0^{th} - 99^{th}$ percentile range of known community sizes of a dataset. These subgraphs follow neither the observed community size distribution, nor any embedded structural patterns of communities. However, they can overlap with a known community. We extract features for all known communities and 10k random subgraphs in the Amazon dataset, and present them in Figures 3(c) and 3(a).

Each row in Figures 3(c) and 3(a) represents a feature vector of a known community subgraph and a random subgraph respectively. That is, each row is like the color scaled feature

vector in Figure 2. Each column represents a labeled edge type. The value in a cell represents the probability of that labeled edge type in that subgraph. K-Means clustering is used to cluster the feature vectors into 5 clusters. Known communities and random subgraphs are clustered and ordered similarly for ease of comparison. Different clusters are separated by a line.

We thus make the following observations:

- **Certain types of feature vectors are more common in communities, while others are common in random subgraphs.** For example, feature vectors like those seen in the first(top) cluster for communities are over represented in random subgraphs, while those in the last(bottom) one are underrepresented. Therefore, a subgraph with a feature vector similar to the first cluster is less likely to be a community, while one similar to the last cluster is more likely to be a community.
- **Patterns in community feature vectors change within and across graph datasets.** Clearly, more than one type of pattern can be seen in the communities within one graph dataset. In many clusters most of the edges are of only a few edge types (color scaled as red), and they change per cluster. This shows a clear and changing connection or interaction bias between node labels. Similar patterns are also observed in other datasets. This lends weight to the idea of subtle information being embedded in community structure, and dataset specific learning for community detection.
- **This approach can identify cliques without explicitly trying to.** Nodes in clique-like communities are likely to have the same node label (say A) owing to them having similar distributions of (very high) Jaccard similarity scores. As a result the feature vectors of all clique-like communities will contain a dominant edge type (A, A) as most edges in it will connect nodes with the label A.

We now introduce some terms that we use throughout the paper. We refer to the centroid of a cluster of community feature vectors as a **subgraph pattern** SP_i, and high probability edge types within a subgraph pattern as **dominant edge types**. The **support of** SP_i is the fraction communities that are clustered together under SP_i. The support of SP_i in a set of known communities, random subgraphs, or detected communities is denoted by $Supp_K(SP_i)$, $Supp_R(SP_i)$, and $Supp_C(SP_i)$ respectively. Note that the centroid represented by SP_i does not change, only its support in each case.

D. Problem Statement

Our analysis till here shows that communities in different datasets follow different size distributions, that some patterns are more characteristic of communities (vs. random subgraphs) and these patterns vary by dataset. Therefore a semi-supervised community detection algorithm that builds upon a training set of communities should extract subgraphs similar to the training set in these aspects.

Consider a graph $G(N, E)$, a set of known communities $K = \{k_1, k_2, ..., k_n\}$, and a function $F(k)$ that represents a community k in some feature space. Then, for each known community $k_i \in K$, we wish to extract a set of communities $C_i = \{c_{i,1}, c_{i,2}, ..., c_{i,m}\}$ which satisfy the following constraints:

1) $|C_i| \approx m, \forall i \in [1...n]$
2) $minimize(||F(c_{i,j}) - F(k_i)||), \forall c_{i,j} \in C_i$
3) $minimize(|||c_{i,j}| - |k_i|||), \forall c_{i,j} \in C_i$

These conditions ensure equal representation of each training community's pattern in the extracted communities (constraint 1), as well as similarity between the training and extracted communities in terms of feature patterns (constraint 2) and size (constraint 3).

Based on our observations in Section III-C, we rewrite these constraints for a set of subgraph patterns $SP = \{SP_1, ..., SP_{n'}\}$, and the feature space defined in Section III-A. For each subgraph pattern SP_i, we wish to find a set of communities $C_i = \{c_{i,1}, c_{i,2}, ..., c_{i_{m'}}\}$ that satisfy the following criteria:

$$min(||Supp_C(SP_i) - Supp_K(SP_i)||), \forall i \in [1...n'] \quad (1)$$

$$min(||F(c_{i,j}) - F(SP_i)||), \forall c_{i,j} \in C_i \quad (2)$$

$$min(||pmf(sizes(C_i)) - pmf(sizes(SP_i))||) \quad (3)$$

Where C is the set of all extracted communities, $sizes$ represents the size of communities in C_i or SP_i, $||-||$ is some measure of dissimilarity depending on context (L_1, KL-divergence, etc.), and pmf is the probability mass function.

IV. Method: Bespoke

Bespoke proceeds in the following manner: For a given graph G, all nodes are assigned one of the N_l labels based on their local topological features. Next, the training communities are used to extract N_p subgraph patterns. Once the subgraph patterns have been extracted, all the nodes in G are scored based on how closely their neighborhoods match each subgraph pattern. This helps identify parts of the graph where communities are more likely to occur. This score is then used by an expansion algorithm to pick seed nodes, and grow communities that match the size distribution of communities in the training set.

Term	Definition
N_l	Number of node labels
N_p	Number of subgraph patterns
SP_i	A subgraph pattern
$F(subgraph)$	Function to get feature vector for a subgraph
$Supp_X(SP_i)$	Support for SP_i in set of subgraphs X
D_{SP_i}	Community size distribution associated with SP_i
$Scores_{SP_i}$	Scores of all nodes under SP_i
N_{find}	Number of communities to find

TABLE II
TABLE OF NOTATIONS

A. Training

1) Subgraph Patterns, and Size Information: Once the nodes have been labeled (Section III-B), features for the training communities are generated (Section III-A). The community features are then clustered using K-Means (K=N_p). We use each cluster center as subgraph pattern SP_i. Each subgraph pattern is associated with a distribution of community sizes D_{SP_i} based on the communities in that cluster.

2) Scoring Nodes based on Subgraph Patterns: Each node is assigned multiple scores, one for each subgraph pattern in the training set. The closer the distribution of edge types in its neighborhood is to a given subgraph pattern, the higher its score for that subgraph pattern. Furthermore, the score is biased towards subgraph patterns with dominant edge types as they show a clear interaction/connection bias between roles. Given a subgraph pattern SP_i, the score of a node n_j is computed in two passes.

Pass 1: The node is scored based on the distribution of edges connected to the node. If the subgraph pattern and neighborhood of the node both have the same dominant edge types, then the node get a higher score.

$$s'(n_j, SP_i) = \frac{\sum_{n_k \in NB(n_j)} P(SP_i, l(n_j), l(n_k))}{deg(n_j)}$$

Where $P(SP_i, a, b)$ is the probability of edges of type (a, b) in subgraph pattern SP_i. $NB(n_j)$ is the set of n_j's neighbors, and $l(node)$ is the label of a node. The score's range is [0,1].

Pass 2: The second pass adds to $s'(n_j, SP_i)$ a degree weighted average of its neighbors' scores. This is similar to heat or probability diffusion. Therefore the node's final score under SP_i is:

$$s(n_j, SP_i) = s'(n_j, SP_i) + \frac{\sum_{n_k \in NB(n_j)} s'(n_k, SP_i) deg(n_k)}{\sum_{n_k \in NB(n_j)} deg(n_k)}$$

For N_p subgraph patterns, each node will have N_p different scores. Instead of averaging the N_p scores, each score is kept and used separately. A node is used to grow communities for only those patterns for which it is a good seed ($s(n_j, SP_i) >= th$).

B. Community Extraction

The algorithm takes the following as input: A graph G, a number of communities to find N_{find}, the list of subgraph patterns SP, their supports in known communities $Supp_K$, community size distribution for each subgraph pattern D_{SP}, and node scores $Scores_{SP}$ that contains node scores under each subgraph pattern.

The algorithm initializes C_{found} to an empty list, and then iteratively adds a community to it till N_{find} communities have been found. Each iteration starts by picking a subgraph pattern, where probability of picking SP_i is proportional to its support in the training set(constraint 1). Next, the algorithm picks a target size T for the community at random from D_i. D_i is the size distribution of communities in SP_i's cluster (constraint 3). A suitable seed node S is picked by the $pick\_seed$ algorithm based on T and $Scores_i$. With the seed and target size determined, $BFExpand$ uses a breadth first expansion to pick T nodes for the community.

1) pick_seed: The Seed Selection Algorithm: Algorithm 2 is used to search for a seed node with a high score, and degree close to T from which to grow the community. $Scores_i$ contains information about the degree and score of each node under SP_i. The function $find\_best\_seed$ starts by setting the required degree for the seed node to be $T-1$. If nodes with that degree are found in $Scores_i$, then the node with the highest score is selected. If no node with that degree is found, the algorithm searches for nodes with degree up to $(T-1)+\epsilon$, where ϵ is a small value. If still no suitable seed node is found, it returns $null$ and a random node is picked as a seed.

2) BFExpand: Given a graph G, a seed node S, and a target size T, $BFExpand$ expands outward from S in a breadth first manner till T unique nodes have been visited. The T visited nodes are returned as the community grown from seed S. Methods like simulated annealing or random walks can be used to grow communities that match the features of SP_i from a seed. However, we found in our experiments that breadth first expansion performed just as well, if not better, and faster than other methods considered.

Algorithm 1 Community Detection

Input: $G, N_{find}, SP, Supp_K, D_{SP}, Scores_{SP}$
Output: List of communities, C_{found}

```
C_found ← []
while len(C_found) < N_find do
    SP_i, Scores_i, D_i = pick_pat(SP, Supp_K, Scores_SP, D_SP)
    T = pick_target_size(D_i)
    S = pick_seed(T, Scores_i)
    C_new = BFExpand(S, T, G)
    C_found.append(C_new)
end while
```

Algorithm 2 $pick\_seed$

Input: $T, Scores_i$
Output: Seed, S

```
S ← find_best_seed(T, Scores_i)
if S == null then
    S ← pick_random(Scores_i)
end if
pop(Scores_i, S)
```

V. Experiments and Results

As noted in Section III-C, clique-like communities should exhibit a distinct pattern in the feature space used by Bespoke, and should be easily detected. We therefore generate synthetic graphs (N∼10k, E∼100k) with 500 assortative communities based on a stochastic block model(SBM) with community sizes based on a power law distribution. We refer to this dataset as **SBM**. This dataset is used as a sanity check for Bespoke's performance on graphs with clique like communities. We limit our analysis of this dataset to quality of communities detected ($F_1 Score$), while the real world datasets are used for further examining Bespoke's performance characteristics.

A. Community Detection Algorithms

BigClam and **RCJoint:** Modularity based community detection algorithms [6], [7].

CESNA [3], which uses node-meta data and graph structure for community detection. We run CESNA on the Twitter and Facebook datasets as node meta-data is available for only those two datasets.

ComE[1], A community detection algorithm based on node embeddings [8].

Must-link: A semi-supervised approach based on [9]. For this approach, we run BigClam on graphs that have been modified by adding extra edges between nodes in the same community. These serve as additional constraints to the BigClam algorithm in a fashion similar to that proposed in [9]. Results for this setup are referred to as "BCA" (BigClam-Assisted) throughout the evaluation.

Random: A straw man baseline algorithm extracts random subgraphs from a given graph. The sizes of the subgraphs are pick uniformly at random in the range [0, 100].

Bespoke-SZ: A version of Bespoke where only community size distribution information is used for detection. This algorithm picks target community sizes (T) based on sizes of communities in a training set. The seed node is picked in a way similar to Bespoke but based only on node degree. A community is then grown from that seed using $BFExpand$.

Bespoke: We implement[2] Bespoke using Python 2.7, and open source python libraries [10]–[12]. The number of training communities, N_{train}, is set to 100 for all datasets except for Facebook and SBM, for which it is set to 20 as they have only a few known communities (191 and 500 respectively). The training set is chosen at random from the set of all communities, and the remaining are used for evaluating its performance. N_l and N_p are fixed at 4 and 5 respectively for all datasets. The quality of detected communities is computed using $F_1Scores$ as described by Yang *et al.* [3].

All algorithms (except CESNA) are set to detect 1500 communities for the Facebook and SBM dataset because of their smaller size, and 50k for all other datasets. CESNA performed best when the "auto" setting for number of communities to detect was used, and the same is reported. The experiments are repeated 20 times and the average $F_1Scores$ for each algorithm on all datasets are reported in Table III.

B. Insights into Performance

We see from Table III that BigClam-Assisted (BCA) improves over the performance of BigClam, as expected. However the improvement is often marginal. This is because the guidance/additional edges added to a graph is not significant in many cases. For example, in all datasets except Facebook, Bespoke trained on communities that contained <0.5% of the complete graph's nodes and edges. This meant that BCA got a small number of additional constraints in to work with, which can explain the marginal gain in most cases. Additionally, this guidance gets spread over thousands of communities. The Facebook dataset is an exception, as it has fewer and larger communities overall(Figure 1). This means that the 20 communities over which Bespoke trains on cover a significant portion of the graph (∼15% of the nodes and edges). This gives BCA a significant amount of guidance per community for that dataset. The ComE algorithm at times performs worse than the baseline random partitioning algorithm.

Method	**Amazon**	**DBLP**	**YT**	**TW**	**FB**	**SBM**
RCJoint	0.42	0.43	0.11	0.37	0.37	0.73
BigClam	0.40	0.41	0.13	0.31	0.32	0.65
BCA	0.42	0.43	**0.13**	0.32	0.38	0.64
ComE	0.40	0.38	–	0.23	0.27	0.67
CESNA	–	–	–	0.34	0.36	–
Rand.	0.22	0.26	0.09	0.30	0.35	0.61
Bespoke-SZ	0.36	0.45	0.11	0.39	0.40	0.73
Bespoke	**0.51**	**0.51**	0.12	**0.40**	**0.44**	**0.81**

TABLE III
$F_1Scores$ OF COMMUNITIES DETECTED THE BY ALGORITHMS OVER THE 5 GRAPH DATASETS. META DATA FOR CESNA IS ONLY AVAILABLE FOR FACEBOOK AND TWITTER DATASETS.

As seen in Table III, Bespoke outperforms all unsupervised and semi-supervised approaches across all datasets except BCA on YouTube. It even outperforms CESNA which uses node meta-data. Finally, as expected, Bespoke performs well on the SBM dataset which has clique-like communities.

Bespoke has significant gain over other approaches for the Amazon and DBLP datasets, while not much for the YouTube dataset. The reason lies partially in the difference in patterns found in random subgraphs and communities in each dataset. The more different the patterns are, the greater the gain. Figures 3(c) and 3(a) show that the support for most subgraph patterns is extremely different in known communities and random subgraphs in the Amazon dataset. Similar trends are seen in the DBLP dataset. However, the YouTube dataset presents a lower degree of mismatch. We also observed that communities in that dataset are star shaped and centered around high degree nodes with (similarly labeled) low degree nodes as neighbors. That makes identifying communities particularly challenging for all methods.

The difference in the performance of the random partitioner, Bespoke-SZ, and Bespoke is especially interesting as the primary difference between these approaches is how the community sizes and seed nodes are chosen. We see from the F_1scores that **knowing the size distribution of communities results in improvement over the random partitioner in all cases**. Further including **structure/seed selection information once again significantly increases the F_1scores**.

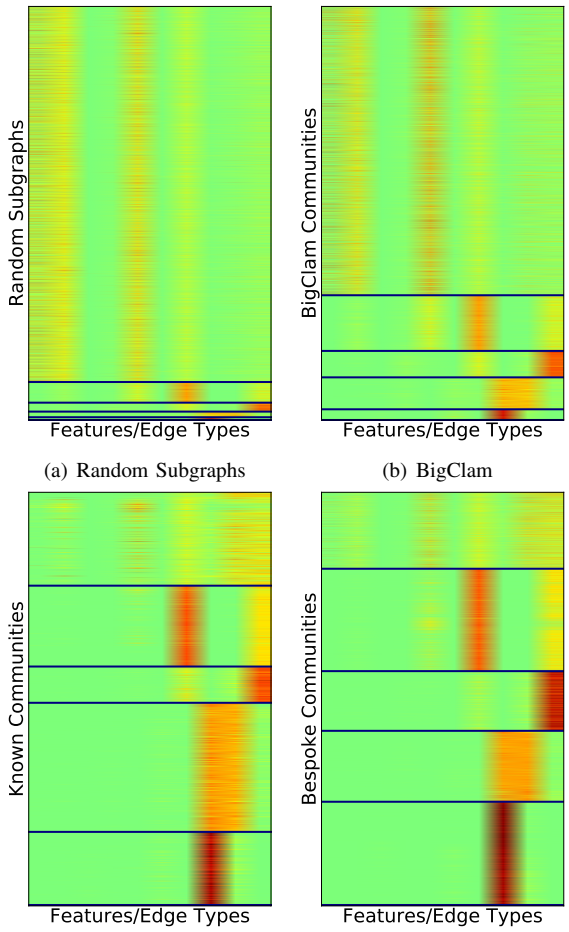
C. Patterns in Data

In Figure 3 we plot the features ($N_l = 4, N_p = 5$) of the known communities, randomly selected subgraphs, and those extracted by BigClam, and Bespoke. Figure 3(c) shows all known communities clustered into 5 patterns. Randomly selected subgraphs do not show many of those patterns and over represent others as seen in Figure 3(a). The patterns seen in communities detected by BigClam in Figure 3(b) resemble the ground truth more than the random subgraphs, but many patterns are still under or over represented. Finally, the communities detected by Bespoke show community feature patterns that closely match those of the ground truth in terms of support and similarity of features. **This shows that**

[1]https://github.com/andompesta/ComE

[2]https://github.com/abaxi/bespoke-icdm18

Bespoke minimizes the conditions outlined in Section III-D. Consequently, it also has the highest F_1Score.



(a) Random Subgraphs (b) BigClam

(c) Known Communities (d) Bespoke

Fig. 3. Patterns in random subgraphs, known communities, and communities extracted by various algorithms for the Amazon dataset.

VI. Related Work

Constraint based community detection approaches incorporate the must/must-not link constraint in to an adjacency matrix [1], [9]. It adds weights between nodes that are known to be in the same community, and then performs non-negative matrix factorization to get communities. These works take as input the local must-link/must-not-link constraints and penalties. In contrast, our approach can finds patterns in known communities that can be generalized over the whole graph.

In some cases metadata about the nodes or edges in a graph is available, which can be helpful for community detection. Yang et al. propose CESNA, which follows the idea that connected nodes with similar attributes are more likely to be in the same community [3]. The work presented here does not rely on metadata, and aims to learn patterns in communities that go beyond clustering nodes with similar metadata.

Node and graph embeddings have been used for problems like multi-label classification, and community detection [8], [13]. For example, ComE [8] generates node embeddings, community embeddings, and identifies communities in a joint optimization process. Most such techniques cluster the node embeddings in order to form communities. Although the embeddings take local features into account, nodes in different parts of the graph can be in the same community. This leads to poor detection accuracy as seen in Table III which shows ComE performing worse than the random algorithm.

VII. Conclusion

In this paper we have presented a new way to look at the problem of community detection, one that leverages size and structural information of known communities. We defined concise and meaningful ways to model the structure and size distribution of communities, and demonstrate that patterns in communities clearly differ from those in random subgraphs in our model's feature space. We also presented a way to easily incorporate these patterns into a community detection algorithm. Our results show that information extracted from a small set of known communities can help exceed the performance of sophisticated modularity based approaches in terms of runtime and quality of communities extracted.

References

[1] E. Eaton and R. Mansbach, "A spin-glass model for semi-supervised community detection." in *AAAI*. Citeseer, 2012.

[2] D. Hric, R. K. Darst, and S. Fortunato, "Community detection in networks: Structural communities versus ground truth," *Physical Review E*, 2014.

[3] J. Yang, J. McAuley, and J. Leskovec, "Community detection in networks with node attributes," in *Data mining (ICDM), 2013 ieee 13th international conference on*. IEEE, 2013.

[4] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[5] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li, "Rolx: structural role extraction & mining in large graphs," in *Proceedings of the 18th ACM SIGKDD*. ACM, 2012.

[6] J. Yang and J. Leskovec, "Overlapping community detection at scale: a nonnegative matrix factorization approach," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, 2013.

[7] Y. Ruan and S. Parthasarathy, "Simultaneous detection of communities and roles from large networks," in *Proceedings of the second ACM conference on Online social networks*. ACM, 2014.

[8] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria, "Learning community embedding with community detection and node embedding on graphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 2017.

[9] Z.-Y. Zhang, "Community structure detection in complex networks with partial background information," *EPL (Europhysics Letters)*, 2013.

[10] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: a structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[11] E. Jones, T. Oliphant, and P. Peterson, "{SciPy}: open source scientific tools for {Python}," 2014.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.