

# Progressive Clustering of Networks Using Structure-Connected Order of Traversal

Dustin Bortner<sup>1</sup>, Jiawei Han<sup>2</sup>

Department of Computer Science

University of Illinois at Urbana-Champaign

<sup>1</sup>bortner1@illinois.edu

<sup>2</sup>hanj@illinois.edu

**Abstract**—Network clustering enables us to view a complex network at the macro level, by grouping its nodes into units whose characteristics and interrelationships are easier to analyze and understand. State-of-the-art network partitioning methods are unable to identify hubs and outliers. A recently proposed algorithm, SCAN, overcomes this difficulty. However, it requires a minimum similarity parameter  $\varepsilon$  but provides no automated way to find it. Thus, it must be rerun for each  $\varepsilon$  value and does not capture the variety or hierarchy of clusters. We propose a new algorithm, SCOT (or Structure-Connected Order of Traversal), that produces a length  $n$  sequence containing all possible  $\varepsilon$ -clusterings. We propose a new algorithm, HintClus (or Hierarchy-Induced Network Clustering), to hierarchically cluster the network by finding only best cluster boundaries (not agglomerative). Results on model-based synthetic network data and real data show that SCOT’s execution time is comparable to SCAN, that HintClus runs in negligible time, and that HintClus produces sensible clusters in the presence of noise.

## I. INTRODUCTION

Large real-world networks, contain pockets of similar nodes called *clusters*. Network clustering enables us to view complex networks at the macro level, by grouping the raw network into units whose characteristics and interrelationships are easier to analyze and understand. For example, social networks may be organized into social groups, and complex biological networks may be grouped into modules to create a biological map.

Spectral partitioning methods, such as normalized cut [1], have been popularly applied to network clustering, but these methods use a cutting hyperplane and cannot be applied recursively to find the number of partitions [2]. More recently, agglomerative hierarchical and probabilistic simulated annealing methods have been proposed [3], [4]. However, these methods cannot identify hubs and outliers, but rather include them in the clusters.

In light of these shortcomings, Xu *et al.* propose a network connectivity-based algorithm, SCAN [5], by extension of a density-based clustering approach, such as DBSCAN [6]. It uses a structural similarity measure and defines clusters using a minimum cluster size  $\mu$  and a minimum similarity  $\varepsilon$ . SCAN runs in average  $O(n)$  time and not only finds clusters, but also identifies outliers and hubs. Nevertheless, there are several difficulties associated with SCAN:

- There is no automated way to find a good  $\varepsilon$ .

- The algorithm must be rerun for each  $\varepsilon$ .
- Epsilon is a global threshold, which implies
  - There are no hierarchical clusters, and
  - there is no variation in cluster subtlety.

We respond with the following contributions, developing:

- The algorithm SCOT, which traverses the network in structure-connected order in average  $O(n)$  time, producing a length- $n$  sequence containing all possible  $\varepsilon$ -clusterings;
- A heap structure (ContigHeap) of size at most  $n$  that stores the cluster boundary hypotheses obtained from SCOT’s output and is built in  $O(n)$  time; and
- The heap-clustering algorithm HintClus, which produces hierarchical clusters (optimal cluster boundaries only, not agglomerative).

## II. STRUCTURE-CONNECTED TRAVERSAL

### A. Motivation

Intuitively, for a pair of nodes, we may define similarity as the ratio of shared neighbors to total neighbors. This is known as *structural similarity*. To capture clusters with differing degrees of structural similarity using a global connectivity threshold parameter, we would need to run SCAN many times with different parameter values. However, there is an unlimited number of possible values, and it is not clear how to combine the results of numerous runs. To solve this problem, we propose an algorithm that produces a length- $n$  sequence containing all possible  $\varepsilon$ -clusterings.

Our algorithm traverses the network nodes according to structural similarity order. We begin at some point and traverse the nodes to a local maximum similarity, where we then iteratively visit all nodes in order of decreasing similarity, essentially generating a spanning tree from *any previously visited node*. When we reach a saddle point, we “spill over”, find the local maximum, and span the nodes in decreasing similarity order.

### B. Structure-Connected Clusters

The main idea of structure-connected clusters is that for each node in a cluster, the node must have at least  $\mu$  neighbors whose structural similarity is at least  $\varepsilon$ . In this subsection, we

briefly survey the concepts of structure-connected cluster; see [5] for a full discussion.

Let  $G = (V, E)$  be a network. For a node  $v \in V$ , the *structure*  $\Gamma(v)$  of  $v$  is the set containing  $v$  and its neighbors. The *structural similarity* between two nodes  $v, w$  is then

$$\sigma(v, w) \equiv \frac{|\Gamma(v) \cap \Gamma(w)|}{\sqrt{|\Gamma(v)||\Gamma(w)|}}.$$

For a node  $v$ , the set of neighbors having structural similarity greater than  $\varepsilon$  forms the  $\varepsilon$ -neighborhood  $N_\varepsilon(v)$  of  $v$ . If  $|N_\varepsilon(v)| \geq \mu$ , then  $v$  is considered a *core* node in the cluster. A core node  $v$  is *directly reachable* from a core node  $u$  if  $v \in N_\varepsilon(u)$ . The transitive closure of direct reachability forms a cluster, and we say that the nodes are *structure connected*.

A *clustering* of a network is the set of distinct structure-connected clusters found in a network, given parameters  $\varepsilon, \mu$ . A node cannot belong to more than one cluster. Nodes that do not belong to any cluster are either hubs or outliers.

### C. Structure-Connected Order

A local  $\varepsilon$  is found for each node by determining the minimum structural similarity (*core similarity*) that would make the current node a core node. We then calculate the structural similarity (*reachability similarity*) from the current node to each  $\varepsilon$ -neighbor. At each iteration, we choose to visit the new node with the greatest reachability similarity (Sim) from *any previously visited* node. This ensures that regions of higher structural similarity are spanned before the surrounding regions of lower structural similarity.

---

#### Algorithm 1 SCOT( $G, \hat{\varepsilon}, \mu$ )

---

**Input:** A network  $G = (V, E)$ , optional minimum reachability similarity  $\hat{\varepsilon}$ , minimum interesting cluster size  $\mu$

**Output:** A sequence of nodes in structure-connected order of traversal.

```

1: for all  $v \in V$  do
2:   if  $v.visited$  then continue
3:   EnqueueNeighbors( $G, v, \hat{\varepsilon}, \mu$ )
4:   if  $v.coreSim \neq \text{UNDEFINED}$  then
5:     while  $|visitQueue| > 0$  do
6:        $currNode := visitQueue.Dequeue()$ 
7:       EnqueueNeighbors( $G, currNode, \hat{\varepsilon}, \mu$ )

```

---



---

#### Algorithm 2 EnqueueNeighbors( $G, currNode, \hat{\varepsilon}, \mu$ )

---

```

1:  $epsNeighbors := G.GetEpsNeighbors(currNode, \hat{\varepsilon})$ 
2:  $currNode.visited := \text{true}$ 
3:  $currNode.SetCoreSim(G, currNode, \mu)$ 
4:  $visitedNodes.Append(currNode)$ 
5: if  $currNode.coreSim \neq \text{UNDEFINED}$  then
6:    $visitQueue.Update(epsNeighbors, currNode)$ 

```

---



---

#### Algorithm 3 VisitQueue.Update( $epsNeighbors, currNode$ )

---

```

1: for all  $nbr \in epsNeighbors$  do
2:   if Not  $nbr.visited$  then
3:      $cSim := currNode.coreSim$ 
4:      $newRSim := \min\{cSim, \sigma(currNode, nbr)\}$ 
5:     if  $nbr.reachSim = \text{UNDEFINED}$  then
6:        $nbr.reachSim := newRSim$ 
7:        $visitQueue.Enqueue(nbr, newRSim)$ 
8:   else
9:     if  $newRSim > nbr.reachSim$  then
10:       $nbr.reachSim := newRSim$ 
11:       $visitQueue.SetPriority(nbr, newRSim)$ 

```

---

Our algorithm SCOT is listed in Algorithm 1. The outer loop beginning at line 1 iterates over each node  $v$  in the network, in arbitrary order. EnqueueNeighbors determines whether  $v$  is a core node, sets its core similarity, and adds  $v$  to the sequence of visited nodes. If  $v$  is a core node, its  $CoreSim(v)$ -neighbors (neighbors in  $N_{CoreSim(v)}(v)$ ) are added to the priority queue  $visitQueue$  or their priority therein is updated by calling  $visitQueue.Update$ . The reachability similarity with respect to  $v$  is calculated and used as the priority. This update ensures that even if a node was previously enqueued, the edge with greatest structural similarity from *any previously visited* node is traversed when the node reaches the head of the queue. Returning to SCOT, if  $v$  is a core node, then the while statement at line 5 iteratively spans the network in structure-connected order of traversal, using the priority queue  $visitQueue$ . Note that execution will not return to line 1 of SCOT unless (a)  $\hat{\varepsilon} = 0$  and the graph is not connected or (b)  $\hat{\varepsilon} > 0$  and this causes some node not to be a core node. If this occurs, any nodes already visited will be skipped, and the next node will be chosen arbitrarily. Therefore, the order of node traversal within the connected subgraphs (which is the entire graph if the graph is connected and  $\hat{\varepsilon} = 0$ ) is determined by structure-connected order of traversal, but the order of disconnected subgraphs and the selection of the initial point for each such subgraph is arbitrary.

Consider a network containing three clusters, each of which has an outlier, and a hub, which is connected to all three clusters. If we plot a bar graph of the output of SCOT for such a network with traversal order index on the horizontal axis and reachability similarity on the vertical axis, then the three clusters are clearly discernible as three “humps” in the plot, and the hub and outliers are located in the low regions between the humps.

### D. Representation of Infinite Solutions

The length- $n$  sequence output by SCOT represents the structure-connected clusterings of a network for *every* possible  $\varepsilon$ . This property allows us to easily generate the equivalent  $\varepsilon$ -clusterings of SCAN from the compact output of SCOT.

*Definition 1: CONTIGUOUS INTERVAL.* The interval  $[a, b]$  is a *contiguous interval* with respect to  $\text{Sim}$  iff  $(\exists \varepsilon > 0)(\forall i \in [a, b]) \text{Sim}(i) \geq \varepsilon$  and  $[a, b]$  is maximal. That is,  $(\nexists a' < a)(\forall i \in [a', a]) \text{Sim}(i) \geq \varepsilon$  and  $(\nexists b' > b)(\forall i \in [b, b']) \text{Sim}(i) \geq \varepsilon$ . This is denoted by  $\text{Contig}_\varepsilon(a, b)$ .

A contiguous interval represents a contiguous, maximal region of the SCOT plot above a horizontal slice.

*Theorem 1:* Any contiguous interval  $\text{Contig}_\varepsilon(a, b)$  corresponds one-to-one to a structurally connected cluster. That is,

$$C = \{\eta_i \mid i \in [a - 1, b]\} \wedge \text{Contig}_\varepsilon(a, b) \Leftrightarrow \text{Cluster}_\varepsilon(C)$$

(Proof omitted due to space limit.)

### III. MU-ONLY CLUSTERING

In this section, we find clusters without a global  $\varepsilon$  parameter. The  $\varepsilon$  we find for each cluster is independent of others and appropriate for that cluster. In addition, we obtain hierarchical clusters, without producing many redundant clusters. Our technique relies on two novel concepts: (i) transforming the SCOT sequence into a heap of contiguous subintervals, and (ii) using a divide-and-conquer statistical pruning method to identify the clusters satisfying the boundary criterion by progressing both root-to-leaf and leaf-to-root in a single traversal of the heap.

#### A. Building the Contiguous Subinterval Heap

Theorem 1 states that each contiguous interval in the SCOT sequence is equivalent to a structure-connected cluster. A natural partial order over the set of contiguous intervals is stated in the following definition.

*Definition 2: CONTIGUOUS SUBINTERVAL.* Given a contiguous interval  $\text{Contig}_\varepsilon(a, b)$ , the interval  $[c, d]$  is a *contiguous subinterval* of  $[a, b]$  iff  $[c, d] \subset [a, b]$  and  $(\exists \varepsilon' > \varepsilon) \text{Contig}_{\varepsilon'}(c, d)$ . This is denoted by  $[c, d] \prec [a, b]$ .

Contiguous subintervals can be stored in a tree structure. The partial order makes this structure a heap, which we call a *ContigHeap*. Our stack-based `BuildContigHeap` algorithm builds the *ContigHeap* in  $O(n)$  time. `BuildContigHeap` can be integrated into SCOT by processing the nodes as they are visited. In the next subsection, we prune the heap to contain only the contiguous subintervals that satisfy the cluster boundary criterion.

#### B. Determining Cluster Boundaries

Local  $\varepsilon$  are naturally determined by where the data has a clear boundary, the *cluster boundary*, at which the change in  $\sigma$  has a local extremum. Consider the bar graph of the SCOT output for some network. If the slope of the graph on both sides of a contiguous subinterval is below some threshold, then, in relation to the cluster boundary criterion, the contiguous subinterval is not bounded at a local extremum of  $\Delta\sigma$  and does not represent a cluster boundary. Child clusters are independent of their parent clusters, to a degree proportional to the fraction of their width with respect to

the parent. In the case that contiguous subintervals form a *chain*, a path in the heap along which the widths of the contiguous subintervals change very little, then the  $\Delta\varepsilon$  in the chain compete for local maximum. Chains define the scope for cluster boundary evaluation.

Consider the following weighting scheme to describe the relevance of some ancestor node in determining whether a node is a cluster boundary in the *ContigHeap*. Let  $n_k$  be the node under evaluation. Then the relevance of the parent node  $n_{k-1}$  to  $n_k$  is given by the ratio of the node's width to its parent's width,  $w_k/w_{k-1}$ . If the ratio is high, then  $n_{k-1}$  is strongly relevant in determining whether  $n_k$  is a cluster boundary; otherwise,  $w_{k-1}$  is less relevant to  $n_k$ . Define the relevance of some ancestor node  $n_{k-d}$  with respect to  $n_k$  as

$$\text{Rel}(w_{k-d}, w_k) \equiv \frac{w_k}{w_{k-1}} \frac{w_{k-1}}{w_{k-2}} \dots \frac{w_{k-d+1}}{w_{k-d}} = \frac{w_k}{w_{k-d}}$$

for  $d < k$ . Then the relevance of  $n_{k-d}$  to  $n_k$  is the product of the pairwise relevances of all intervening node pairs in the path from  $n_k$  to  $n_{k-d}$ . Thus, if all intervening node pairs have high pairwise relevance, then they form a chain and  $n_{k-d}$  is highly relevant to  $n_k$ . However, if any of the intervening node pairs has a low pairwise relevance, then this breaks the chain, for all following nodes will have diminished relevance.

To satisfy the cluster boundary criterion, we define a cluster boundary as a contiguous subinterval having  $\Delta\varepsilon$  at least one weighted standard deviation away from the weighted mean  $\Delta\varepsilon$ , calculated during progression in both the root-to-leaf and leaf-to-root directions, using the relevances as the weights. In the definition of  $\text{Rel}$ , we see that the iterated products collapse to the ratio of  $w_k/w_{k-d}$ . Thus the weighted mean and weighted standard deviation in the root-to-leaf direction are

$$m_k \equiv \frac{\sum_{i=1}^{k-1} \frac{w_k}{w_i} \Delta\varepsilon_i}{\sum_{i=1}^{k-1} \frac{w_k}{w_i}} = \frac{\sum_{i=1}^{k-1} \frac{1}{w_i} \Delta\varepsilon_i}{\sum_{i=1}^{k-1} \frac{1}{w_i}} \quad \text{and}$$

$$s_k \equiv \sqrt{\frac{\sum_{i=1}^{k-1} \frac{w_k}{w_i} (\Delta\varepsilon_i - m_k)^2}{\sum_{i=1}^{k-1} \frac{w_k}{w_i}}} = \sqrt{\frac{\sum_{i=1}^{k-1} \frac{1}{w_i} (\Delta\varepsilon_i - m_k)^2}{\sum_{i=1}^{k-1} \frac{1}{w_i}}}.$$

Because  $w_k$  divides out of  $m_k$  and  $s_k$ , we can use the running zeroth, first, and second moments to calculate  $m_k$  and  $s_k$  efficiently for each node in the *ContigHeap*. A similar argument applies to calculations in the leaf-to-root direction.

Our algorithm `HintClus`, which determines the cluster boundaries by progressing in both directions, is given in Algorithm 4. We call `HintClus`( $\mu, 0, \dots, 0$ ) on the root node of the *ContigHeap*, and the algorithm prunes unwanted contiguous subintervals from the *ContigHeap*.

## IV. EXPERIMENTAL RESULTS

### A. Computational Complexity

We run trials of both SCOT and SCAN on generated datasets of three popular network models: Erdős-Rényi random

---

**Algorithm 4** CHNode.HintClus( $\mu, s_0, s_1, s_2, \varepsilon_{par}, \& cs_0, \& cs_1, \& cs_2$ )

---

**Input:** Minimum interesting cluster size  $\mu$ , root-to-leaf weighted moments  $s_i, \varepsilon$  of parent node

**Output:** Leaf-to-root weighted moments  $cs_i$  by reference

```

1:  $\Delta\varepsilon := \varepsilon - \varepsilon_{par}$ 
2:  $w := b - a$ 
3:  $active := (w + 1 \geq \mu)$ 
4: if  $active$  and  $s_0 \neq 0$  then
5:    $m := s_1/s_0$ 
6:    $s := (1/s_0)\sqrt{s_0s_2 - s_1^2}$ 
7:    $active := (\Delta\varepsilon - m \geq s)$ 
8:  $s_0 := s_0 + 1/w$ 
9:  $s_1 := s_1 + \Delta\varepsilon/w$ 
10:  $s_2 := s_2 + \Delta\varepsilon^2/w$ 
11:  $ts_0 := ts_1 := ts_2 := 0$ 
12: for all  $child$  in  $children$  do
13:    $child.HintClus(\mu, s_0, s_1, s_2, \varepsilon, ts_0, ts_1, ts_2)$ 
14: if  $active$  and  $ts_0 \neq 0$  then
15:    $m := ts_1/ts_0$ 
16:    $s := (1/ts_0)\sqrt{ts_0ts_2 - ts_1^2}$ 
17:    $active := (\Delta\varepsilon - m \geq s)$ 
18:  $cs_0 := cs_0 + ts_0 + w$ 
19:  $cs_1 := cs_1 + ts_1 + \Delta\varepsilon \cdot w$ 
20:  $cs_2 := cs_2 + ts_2 + \Delta\varepsilon^2 \cdot w$ 

```

---

graph, Watts-Strogatz small-world, and Barabási-Albert scale-free models. For each model, we run ten trials with increasing numbers of nodes (and edges). We repeat this for different values of  $\varepsilon$ . The results show that SCOT competes with SCAN in terms of execution time; SCOT has the same computational complexity as SCAN. Further, we find that SCOT runs faster than SCAN for the BA model, which is encouraging, since we observe that the large real-world datasets resemble this model. Finally, we see that BuildContigHeap and HintClus run in relatively negligible time.

### B. Cluster Evaluation

To evaluate the clustering quality of HintClus, we use the Enron email dataset [7]. It contains 517,431 email messages from 150 employees of Enron Corp. We extract valid From and To addresses to yield a network of 56,985 nodes (email addresses) and 219,368 edges (emails).

1) *General Clustering Behavior:* For  $\mu = 30$  and  $\hat{\varepsilon} = 0$ , SCOT executes in 68 s ( $\mu = 30$  to reduce noise; we find similar execution times for  $\mu = 2$  and  $\mu = 5$ ). BuildContigHeap and HintClus execute in less than the minimum time resolution. In the SCOT sequence bar graph, nodes are distributed with a fat tail, along which we encounter spikes whose base  $\varepsilon$  decreases with the tail. Close inspection of the HintClus clusters reveals correspondence to sharp changes

in  $\sigma$ , relative to surrounding regions. In terms of  $\varepsilon$ , the frequency of clusters within a path of the ContigHeap is commensurate with the length of the chain. HintClus captures hierarchy and variety in the clusters and finds only best cluster boundaries, while SCAN cannot feasibly produce any variety of hierarchical clusters.

2) *Clusters Discovered:* To illustrate the legitimacy of the clusters discovered by HintClus, we select five small, non-overlapping clusters and analyze the relationship of the nodes using the contents of the emails sent between the users in that cluster. These examples illustrate that the clusters have legitimate intracluster similarity and intercluster dissimilarity.

## V. CONCLUSION

By traversing the network in structure-connected order, SCOT is able to represent all of SCAN's clusterings for any value of  $\varepsilon$  in one length- $n$  sequence. We presented the algorithm HintClus, which produces hierarchical clusters. Using three popular network models, we compared the computational complexity of SCOT and SCAN, and found them to be identical. With the Enron dataset, we demonstrated the hierarchical and varied clusters produced by HintClus, and explained why it is infeasible to replicate such results with SCAN. In all experiments BuildContigHeap and HintClus ran in negligible time.

## ACKNOWLEDGEMENT

The work was supported in part by the U.S. National Science Foundation grants IIS-08-42769 and BDI-05-15813, Office of Naval Research (ONR) grant N00014-08-1-0565, and the Air Force Office of Scientific Research MURI award FA9550-08-1-0265. Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

## REFERENCES

- [1] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 888–905, 2000. [Online]. Available: <http://dx.doi.org/10.1109/34.868688>
- [2] A. Nagai, "Inappropriateness of the criterion of k-way normalized cuts for deciding the number of clusters," *Pattern Recogn. Lett.*, vol. 28, no. 15, pp. 1981–1986, 2007.
- [3] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, no. 6, p. 066111, Dec 2004.
- [4] R. Guimera and L. A. N. Luis, "Functional cartography of complex metabolic networks," *Nature*, vol. 433, no. 7028, pp. 895–900, February 2005. [Online]. Available: <http://dx.doi.org/10.1038/nature03288>
- [5] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, "Scan: a structural clustering algorithm for networks," in *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2007, pp. 824–833.
- [6] M. Ester, H.-P. Kriegel, S. Jörg, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," pp. 226–231, 1996. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.1980>
- [7] A. Cohen, "Enron email dataset," March 2004. [Online]. Available: <http://www.cs.cmu.edu/~enron/>