Noname manuscript No.
(will be inserted by the editor)

Reducing Large Graphs to Small Supergraphs: A Unified Approach

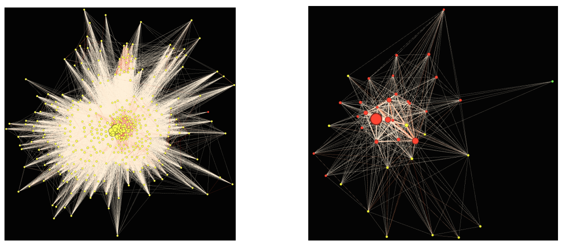
Yike Liu, E-mail: yikeliu@umich.edu
Tara Safavi, E-mail: tsafavi@umich.edu
Neil Shah, E-mail: neilshah@cs.cmu.edu
Danai Koutra, E-mail: dkoutra@umich.edu

Abstract Summarizing a large graph with a much smaller graph is critical for applications like speeding up intensive graph algorithms and interactive visualization. In this paper, we propose CONDENSE (CONditional Diversified Network Summarization), a Minimum Description Length-based method that summarizes a given graph with approximate "supergraphs" conditioned on a set of diverse, predefined structural patterns. CONDENSE features a unified pattern discovery module and a set of effective summary-assembly methods, including a powerful parallel approach, K-STEP, that creates high-quality summaries not biased toward specific graph structures. By leveraging CONDENSE's ability to efficiently handle overlapping structures, we contribute a novel evaluation of seven existing clustering techniques by going beyond classic cluster quality measures. Extensive empirical evaluation on real networks in terms of compression, runtime, and summary quality shows that CONDENSE finds 30-50% more compact summaries than baselines, with up to 75-90% fewer structures and equally good node coverage.

1 Introduction

In an era of continuous generation of large amounts of data, summarization techniques are becoming increasingly crucial to help abstract away noise, uncover patterns, and inform human decision processes. Here we focus on the summarization of *graphs*, which are powerful structures that capture a number of phenomena, from communication between people [5, 31, 33] to links between webpages [27], to interactions between neurons in our brains [41, 46]. In general, graph summarization or coarsening approaches [36] seek to find a concise representation of the input graph that reveals patterns in the original data, while usually preserving specific network properties. As graph summaries are application-dependent, they can be defined with respect to various aspects: they can preserve specific structural patterns, focus on some entities in the network, preserve the answers to a specific set of queries, or maintain the distributions of some graph properties. Graph summarization leads to the reduction of data volume, speedup of graph algorithms, improved storage and query time, and interactive visualization. Its major challenges are in effectively handling the volume and complexity of data, defining the interestingness of patterns, evaluating the proposed summarization techniques, and capturing network structural changes over time. The graph

University of Michigan · Carnegie Mellon University



(a) Prior work [29] (b) Our method: CONDENSE-STEP

Fig. 1: **CONDENSE generates simpler and more compact supergraphs than baselines**. Yellow, red, and green nodes for stars, cliques, and bipartite cores, respectively.

mining community has mainly studied summarization techniques for the structure of static, plain graphs [10,40] and to a smaller extent, methods for attributed or dynamic networks [47].

Our method, CONDENSE or CONditional Diversified Network Summarization, summarizes the structure of a given large-scale network by selecting a small set of its most informative structural patterns. Inspired by recent work [29,40], we formulate graph summarization as an information-theoretic optimization problem in search of local structures that collectively minimize the description of the graph. CONDENSE is a unified, edge-overlap-aware graph summarization method that summarizes a given graph with approximate "supergraphs" conditioned on diverse, predefined structural patterns. An example is shown in Figure 1, where the (super)nodes in Figure 1b correspond to *sets of nodes* in the original graph. Specifically, the predefined patterns include structures that have well-understood graph-theoretical properties and are found in many real-world graphs [4,15,27,42]: cliques, stars, bipartite cores, chains, and patterns with skewed degree distribution.

Our work effectively addresses three main limitations of prior summarization work [28, 29], namely: (i) its heavy dependence on the structural pattern discovery method and intrinsic tendency, or *bias*, to select star-like structures in the final summary; (ii) its inability to handle edge-overlapping patterns in the summary; and (iii) its dependence on the order in which candidate structures are considered for the final summary. Our proposed unified approach effectively handles these issues and results in robust, compact summaries with $5-10\times$ fewer structural patterns (or supernodes) and up to 50% better compression.

CONDENSE consists of three modules that address the aforementioned shortcomings. (i) A unified structural pattern discovery module leverages the strengths of various popular graph clustering methods (e.g., LOUVAIN [8], METIS [26]) to address the biases toward specific structures per clustering method; (ii) A Minimum Description Length-based (MDL) formulation with a penalty term effectively minimizes redundancy in edge coverage by the structural patterns included in the summary. This term is paramount when the candidate structural patterns have significant edge overlap, such as in the case of our unified structure discovery module; (iii) An iterative, multi-threaded, and divide-and-conquer-based summary assembly module further reduces structure selection bias during the summary creation process by being *independent* of the order in which the candidate structural patterns are considered. This parallel module is up to $53\times$ faster than its serial counterpart on a 6-core machine.

Our contributions in this paper are as follows:

- **Approach**: We introduce CONDENSE, an effective unified, edge-overlap-aware graph summarization approach. CONDENSE includes a powerful parallel summary assembly module, K-STEP, that creates compact and easy-to-understand graph summaries with high node coverage and low redundancy.

- **Novel Metric**: We propose a way to leverage CONDENSE as a proxy to compare graph clustering methods with respect to their summarization performance on large, real-world graphs. Our work complements the usual evaluation metrics in the related literature (e.g., modularity, conductance).
- **Experiments**: We present a thorough empirical analysis on real networks to evaluate the summary quality and runtime, and study the properties of seven clustering methods.

For reproducibility, the code is available online at `https://github.com/yikeliu/ConDeNSe`. Next, we present the related work and necessary background.

2 Related Work and Background

Our work is related to (i) graph summarization methods, (ii) compression and specifically MDL, (iii) graph clustering, and (iv) graph sampling. We briefly review each of these topics.

Graph Summarization. Most research efforts in graph summarization [36] focus on plain graphs and can be broadly classified as group-based [32,44], compression-based [10,19,40], simplification-based, influence-based, and pattern-based [12]. Dynamic graph summarization has been studied to a much smaller extent [24,47]. Beyond the classic definition of graph summarization, there are also approaches that summarize networks in terms of structural properties (e.g., degree, PageRank) by automatically leveraging domain knowledge [22,23]. Most related to our work are the ideas of node grouping and graph compression. Built on these ideas, two representative methods, MDL-SUMMARIZATION [40] and VOG [29], are MDL-based summarization methods that compress the graphs by finding near-structures (e.g., (near-) cliques, (near-) bipartite cores). MDL-SUMMARIZATION, which iteratively combines neighbors into supernodes as long as it helps with minimizing the compression cost, includes mostly cliques and cores in the summaries, and has high runtime complexity.

On the other hand, VOG finds structures by employing SLASHBURN [25] (explained below) and hence is particularly biased towards selecting star structures. Moreover, it creates summaries (i.e., lists of structures) using a greedy heuristic on a pre-ordered set of structures (cf. Section 4.3). Unlike these methods, CONDENSE performs ensemble pattern discovery and handles edge-overlapping structures. Furthermore, its summary assembly is robust to the ordering of structures.

MDL in Graph Mining. Many data mining problems are related to summarization and pattern discovery, and, thus, to Kolmogorov complexity [14], which can be practically implemented by the MDL principle [45]. Applications include clustering [11], community detection [9], pattern discovery in static and dynamic networks [29,47], and more.

Graph Clustering. Graph clustering and community detection are of great interest to many domains, including social, biological, and web sciences [6,16,18]. Here, we leverage several graph clustering methods to obtain diversified graph summaries, since each method is biased toward certain types of structures, such as cliques and bipartite cores [8,26,49] or stars [25]. Unlike the existing literature [35] where clustering methods are compared with respect to classic quality measures, we also propose to use CONDENSE as a vessel to evaluate the methods' summarization power. We leverage seven decomposition methods, compared quantitatively in Table 1:

- SLASHBURN [25] is a node reordering algorithm initially developed for graph compression. It performs two steps iteratively: (i) It removes high-centrality nodes from the graph; (ii) It reorders nodes such that high-degree nodes are assigned the lowest IDs and nodes from disconnected components get the highest IDs. The process is repeated on the giant connected component. We use SLASHBURN by identifying structures from the egonet of each high-centrality node, and the disconnected components, as subgraphs.

- **LOUVAIN** [8] is a modularity-based partitioning method for detecting hierarchical community structure. Like SLASHBURN, LOUVAIN is iterative: (i) Each node is placed in its own community. Then, the neighbors j of each node i are considered, and i is moved to j's community if the move produces the maximum modularity gain. The process is applied repeatedly until no further gain is possible. (ii) A new graph is built whose supernodes represent communities, and superedges are weighted by the sum of weights of links between the two communities. The algorithm typically converges in a few passes.
- **SPECTRAL** clustering refers to a class of algorithms that rely on eigendecomposition to identify community structure. We use one such spectral clustering algorithm [20], which partitions a graph by performing k-means clustering on the top-k eigenvectors of the input graph. The idea behind this clustering is that nodes with similar connectivity have similar eigen-scores in the top-k vectors and form clusters.
- **METIS** [26] is a cut-based k-way multilevel graph partitioning scheme based on multilevel recursive bisection (MLRB). Until the graph size is substantially reduced, it first coarsens the input graph by grouping nodes into supernodes iteratively such that the edge-cut is preserved. Next, the coarsened graph is partitioned using MLRB, and the partitioning is projected onto the original input graph G through backtracking. The method produces k roughly equally-sized partitions.
- **HYCOM** [4] is a parameter-free algorithm that detects communities with hyperbolic structure. It approximates the optimal solution by iteratively detecting important communities. The key idea is to find in each step a single community that minimizes an MDL-based objective function given the previously detected communities. The iterative procedure consists of three steps: community candidates, community construction, and matrix deflation.
- **BIGCLAM** [49] is a scalable overlapping community detection method. It is built on the observation that overlaps between communities are densely connected. By explicitly modeling the affiliation strength of each node-community pair, the latter is assigned a nonnegative latent factor which represents the degree of membership to the community. Next, the probability of an edge is modeled as a function of the shared community affiliations. The identification of network communities is done by fitting BIGCLAM to a given undirected network G.

Table 1: Qualitative comparison of the graph clustering techniques included in CONDENSE. Symbols: $n =$ number of nodes, $m =$ number of edges, $k =$ number of clusters/partitions, $t =$ number of iterations, $d =$ average degree, $h(m_h) =$ number of nodes (edges) in hyperbolic structure.

	SLASHBURN [25]	**LOUVAIN [8]**	**SPECTRAL [20]**	**METIS [26]**	**HYCOM [4]**	**BIGCLAM [49]**	**KCBC [37]**
Overlapping Clusters	✔	✘	✘	✘	✔	✔	✔
Cliques	Many	Many	Many	Many	Some	Many	Many
Stars	Many	Some	Some	Some	Many	Some	Some
Bipartite Cores	Some	Few	Many	Some	Some	Few	Few
Chains	Few	Few	Few	Few	Few	Few	Few
Hyperbolic Structures	Few	Few	Few	Few	Many	Few	Few
Complexity	$O(t(m+ +n\log n))$	$O(n\log n)$	$O(n^3)$	$O(m\cdot k)$	$O(k(m+h\log h^2 +hm_h))$	$O(d\cdot n\cdot t)$	$O(t(m+n))$
Summariza-tion Power	Excellent	Very Good	Good	Good	Poor	Good	Poor

- **KCBC** [37] is inspired by the k-cores algorithm [17], which unveils densely connected structures. A k-core is a maximal subgraph for which each node is connected to at least k other nodes. KCBC iteratively removes k-cores starting by setting k equal to the maximum core number (the max value k for which the node is present in the resulting subgraph) across all nodes. Each connected component in the induced subgraphs is identified as a cluster, and is removed from the original graph. The process is repeated on the remaining graph.

Other clustering methods that we considered (e.g., Weighted Stochastic Block Model or WSBM) are not included in CONDENSE due to lack of scalability. For instance, WSBM took more than a week to finish on our smallest dataset.

Graph sampling. Sampling graph nodes and/or edges may be considered an alternative method of graph compression, and as such these techniques relate to graph summarization [7, 21]. Graph sampling techniques have been extensively studied and reviewed [2, 39]. Node sampling methods include sampling according to degree, PageRank score, or substructures like spanning trees. Edge sampling techniques include uniform sampling and sampling according to edge weights or effective resistance [48] to maintain node reachability [3] or the graph spectrum up to some multiplicative error. Graph sampling can also be used to approximate queries with theoretical guarantees.

That said, the fundamental goals of graph sampling and summarization differ. Sampling focuses on obtaining sparse subgraphs that maintain properties of the original input graph, like degree distribution, size distribution of connected components, diameter, or community structure [38]. Unlike graph summarization, sampling is less concerned with identifying succinct patterns or structures that represent the input graph and assist user understanding. Although sampling has been shown to support visualization [43], these methods usually operate on individual nodes/edges instead of collective patterns. Furthermore, the results of graph sampling algorithms may require additional processing for interpretability.

3 CONDENSE: Proposed Model

We formulate the graph summarization problem as a graph compression problem. Let $G(\mathcal{V}, \mathcal{E})$ be a graph with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges, without self-loops. The Minimum Description Length (MDL) problem, which is a practical version of Kolmogorov Complexity [14], aims to find the best model M in a given family of models $\mathcal{M}$ for some observed data $\mathcal{D}$ such that it minimizes $L(M) + L(\mathcal{D}|M)$. In this formulation, $L(M)$ is the description length of M in bits and $L(\mathcal{D}|M)$ is the description length of $\mathcal{D}$ which is encoded by the chosen model M. Table 2 defines the recurrent symbols used in this section.

We consider summaries in the model family $\mathcal{M}$, which consists of all possible permutations of subsets of structural patterns in Ω. One option is to populate Ω with the frequent patterns that occur in the input graph (in a data-driven manner), but frequent subgraph mining is NP-complete and does not scale well. Moreover, even efficient *approximate* approaches are not applicable to unlabeled graphs and can only handle small graphs with a few tens or hundreds of nodes. To circumvent this problem, we populate Ω with five patterns that are common in real-world static graphs [4, 27], correspond to interesting real behaviors, and can (approximately) describe a wide range of structural patterns: stars (*st*), *full* cliques (*fc*), bipartite cores (*bc*), chains (*ch*), and hyperbolic structures with skewed degree distribution (*hs*). Under the MDL principle, any approximate structures (e.g., near-cliques) can be easily encoded as their corresponding exact structures (e.g., *fc*) with some errors. Since many communities have hyperbolic structure [4], which cannot be expressed as a simple composition

Table 2: Major symbols and definitions.

Notation	**Description**
$G(\mathcal{V},\mathcal{E})$, $\mathbf{A}$	graph, and its adjacency matrix
$\mathcal{V}$, $n = \lvert\mathcal{V}\rvert$	node-set and number of nodes of G, resp.
$\mathcal{E}$, $m = \lvert\mathcal{E}\rvert$	edge-set and number of edges of G, resp.
k	# of clusters or communities or patterns
t	# of iterations
h, m_h	size of hyperbolic community, and # of edges in it, resp.
d	average degree of nodes in G
h_{slash}	# of hub nodes to slash per iteration in SLASHBURN
fc, bc, st, ch, hs	full clique, bipartite core, star, chain, hyperbolic structure, resp.
$\lvert fc\rvert, \lvert bc\rvert, \lvert st\rvert, \lvert ch\rvert, \lvert hs\rvert$	number of nodes in the corresponding structure
Ω	predefined set of structural pattern types
M	a model or summary for G
s	structure in M
$\lvert S\rvert$, $\lvert s\rvert$	cardinality of set S and number of nodes in s, resp.
$\lvert\lvert s\rvert\rvert, \lvert\lvert s\rvert\rvert'$	# existing and non-existing edges of $\mathbf{A}$ that s describes
$\mathbf{E}$	error matrix, $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$, where $\oplus$ is exclusive OR
$\mathbf{O}$	edge-overlap penalty matrix
$L(G, M)$	# of bits to describe model M, and G using M
$L(M), L(O), L(s)$	# of bits to describe M, the edge overlap O, and structure s

of the other structural patterns in Ω, we consider this structure separately. Motivated by real-world discoveries, we focus on structures that are commonly found in networks, but our framework is not restricted to them; it can be readily extended to other, application-dependent types of structures as well.

Formally, we address the following problem:

PROBLEM 1. Given a graph G with adjacency matrix $\mathbf{A}$ and structural pattern types Ω, we seek to find the model M that minimizes the encoding length of the graph and the redundancy in edge coverage:

$$L(G, M) = L(M) + L(\mathbf{E}) + L(\mathbf{O}) \qquad (1)$$

where $\mathbf{M}$ is $\mathbf{A}$'s approximation induced by M, $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ is the error matrix to correct for edges that were erroneously described by M, $\oplus$ is exclusive OR, and $\mathbf{O}$ is the edge-overlap matrix to penalize edges covered by many patterns.

Model M induces a supergraph with each $s \in M$ as an (approximate) supernode, and weighted superedges between them. Before we further formalize the task of encoding the model, the error matrix, and the edge-overlap penalty matrix, we provide a visual illustration of our MDL objective.

An Illustrative Example. *Figure 2 shows the original adjacency matrix* $\mathbf{A}$ *of an input graph, which is encoded as (i)* $\mathbf{M}$*, the matrix that is induced by the model* M*, and (ii) the error matrix* E*, which captures additional/missing edges that are not properly described in* M*. In this example, there are 6 structures in the model (from the top left corner to the bottom right corner: a star, a large clique, a small clique, a bipartite core, a chain, and a hyperbolic structure), where the cliques and the bipartite core have overlapping nodes and edges.*

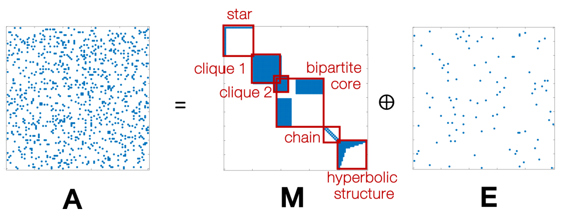


Fig. 2: Illustration of MDL encoding.

3.1 Encoding the Model

Following the literature [29], for an input graph G, to fully describe a model $M \in \mathcal{M}$ that consists of a set of structural patterns s (e.g., stars, hyperbolic structures, and chains), we encode it as $L(M)$, where we use optimal encoding for all its components:

$$L(M) = L_{\mathbb{N}}(|M|+1) + \log_2 \binom{|M|+|\Omega|-1}{|\Omega|-1} + \sum_{s \in M} \big(-\log_2 \Pr(x(s) \mid M) + L(s) \big). \quad (2)$$

In the first term we encode the number of structural patterns in M using Rissanen's optimal encoding for integers (≥ 1) [45]. Then, we transmit the number of patterns per type in Ω by optimally encoding it via an index over a weak number composition. In the third term, for each structure $s \in M$, we encode its type $x(s)$ using optimal prefix codes [13], and its connectivity $L(s)$. For the last term, to capture the real connectivity patterns of each structure s, we introduce the MDL encoding per type of structure in Ω next. As in Equation (2), we optimally encode the various components of each structure (e.g., by using Rissanen's optimal encoding for integers).

- **Stars:** A star consists of a "hub" node connected to two or more "spoke" nodes. We encode it as:

$$L(st) = L_{\mathbb{N}}(|st|-1) + log_2 n + log_2 \binom{n-1}{|st|-1} \quad (3)$$

  where we encode in order the number of spokes, the hub ID (we identify it out of n nodes using an index over the combinatorial number system), and the spoke IDs.
- **Cliques:** A clique is a densely connected set of nodes with:

$$L(fc) = L_{\mathbb{N}}(|fc|) + log_2 \binom{n}{|fc|} \quad (4)$$

  where we encode its number of nodes followed by their IDs.
- **Bipartite Cores:** A bipartite core consists of two non-empty sets of nodes, L and R, which have edges only between them, and $L \cap R = \emptyset$. Stars are a special case of bipartite cores with $|L| = 1$. The encoding cost is given as:

$$L(bc) = L_{\mathbb{N}}(|L|) + L_{\mathbb{N}}(|R|) + log_2 \binom{n}{|L|} + log_2 \binom{n}{|R|}, \quad (5)$$

  where we encode the number of nodes in L and R followed by the node IDs per set.
- **Chains:** A chain is a series of nodes that are linked consecutively–e.g. node-set $\{a, b, c, d\}$ in which a is connected to b, b is connected to c, and c is connected to d. Its encoding cost, $L(ch)$, is:

$$L(ch) = L_{\mathbb{N}}(|ch|-1) + \sum_{i=1}^{|ch|} log_2(n-i+1) \quad (6)$$

  where we encode its number of nodes, and then their node IDs in order of connection.
- **Hyperbolic Structures:** A hyperbolic structure or community [4] has skewed degree distribution which often follows a power law with exponent between -0.6 and -1.5. The encoding length of a hyperbolic structure hs is given as:

$$L(hs) = k + L_{\mathbb{N}}(|hs|) + \log_2 \binom{n}{|hs|} + \log_2(|\mathbf{A}(hs)|) + ||hs||l_1 + ||hs||' l_0 \quad (7)$$

  where we first encode the power-law exponent (using Rissanen's encoding [45] for the integer part, the number of decimal values, and the decimal part), followed by the number of nodes and their IDs. Then, we encode the number of edges in the structure (=$|\mathbf{A}(hs)|$), and use optimal prefix codes, l_0, l_1, for the missing ($||hs||'$) and present ($||hs||$) edges, respectively. Specifically, $l_1 = -\log((||hs||/(||hs|| + ||hs||'))$, and l_0 is defined similarly.

3.2 Encoding the Errors

Given that M is a summary, and $\mathbf{M}$ is only an approximation of $\mathbf{A}$, we also need to encode errors of the model. For instance, a near-clique is represented as a full clique in the model, and, thus, contributes some edges to the error matrix (i.e., the missing edges from the real data). We encode the error $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ in two parts, $\mathbf{E}^+$ and $\mathbf{E}^-$, since they likely follow different distributions [29]. The former encodes the edges induced by M which were not in the original graph, and the latter the original edges that are missing in M:

$$L(\mathbf{E}^+) = log_2(|\mathbf{E}^+|) + ||\mathbf{E}^+||l_1 + ||\mathbf{E}^+||'l_0 \quad (8)$$

$$L(\mathbf{E}^-) = log_2(|\mathbf{E}^-|) + ||\mathbf{E}^-||l_1 + ||\mathbf{E}^-||'l_0 \quad (9)$$

where we encode the number of 1s in $\mathbf{E}^+$ (or $\mathbf{E}^-$), followed by the actual 1s and 0s using optimal prefix codes (as before).

3.3 Encoding the Edge-Overlap Penalty

Several of the graph decomposition methods that we consider (e.g., SLASHBURN, KCBC in Table 1) generate edge-overlapping patterns. The MDL model that we have presented so far naturally handles node overlaps—if two structures consist of the same large set of nodes, only one of the them will be chosen during the encoding cost minimization process, because their combination would lead to higher encoding cost. However, up to this point, the model considers a binary state for each edge: that is, an edge is described by the model M, or not described by it. This can lead to summaries with high redundancy in edge coverage, as we show next with an illustrative example.

To explicitly handle extensive edge overlaps in the graph summaries (which can lead to low node/edge coverage), we add a penalty term, $L(\mathbf{O})$, in the optimization function in Equation (1). We introduce the matrix $\mathbf{O}$, which maintains the number of times each edge is described by M, i.e., the number of selected structures in which the edge occurs. We encode the description length of $\mathbf{O}$ as:

$$L(\mathbf{O}) = \log_2(|\mathbf{O}|) + ||\mathbf{O}||l_1 + ||\mathbf{O}||'l_0 + \sum_{o \in \mathcal{E}(\mathbf{O})} L_{\mathbb{N}}(|o|) \quad (10)$$

where we first encode the number of distinct overlaps, and then use the optimal prefix code to encode the number of the present and missing entries in $\mathbf{O}$. As before, l_0 and l_1 are the lengths of the optimal prefix codes for the present and missing entries, respectively. Finally, we encode the weights in $\mathbf{O}$ using the optimal encoding for integers $L_{\mathbb{N}}$ [45]. We denote with $\mathcal{E}(\mathbf{O})$ the set of non-negative entries in matrix $\mathbf{O}$.

An Illustrative Example. *Let us assume that the output of an edge-overlapping graph clustering method consists of three full cliques: (1) full clique 1 with nodes 1-20; (2) full clique 2 with nodes 11-30; and (3) full clique 3 with nodes 21-40. The encoding that does*

Algorithm 1 CONDENSE

```
 1: Input: graph $G$, parameters of clustering methods in Module A
 2: // Module A: Pattern discovery: Discovery of a diverse set of patterns $P$.
 3: $P$ = SLASHBURN $(G, h_{slash})$ $\cup$ LOUVAIN $(G, \tau)$ $\cup$ SPECTRAL $(G, k)$ $\cup$ METIS $(G, k)$
 4:       $\cup$ HYCOM $(G)$ $\cup$ BIGCLAM $(G)$ $\cup$ KCBC $(G)$          $\{$discussion of parameters in Sec. 5$\}$
 5: //Module B: MDL-based structural pattern identification as full cliques, bipartite cores, stars,
 6: // chains and hyperbolic structures.
 7: for $g \in P$
 8:     for $\omega \in \Omega$
 9:         // e.g., hub identification in star structure $\omega$='st' (Section 4.2)
10:         $r(g, \omega) =$ 'best' representation of $g$ as structure type $\omega$
11:     // $s$: type of structure for pattern $g$ using its best representation $r(g, \omega)$
12:     $s = \arg\min_{\omega \in \Omega} L_{r(g,\omega)}(g, \omega) = \arg\min_{\omega \in \Omega}\{L(\omega) + L(E_\omega^+) + L(E_\omega^-)\}$  $\{$using Eq. (3)-(7)$\}$
13: // Module C: Overlap-aware summary assembly by employing a STEP variant (Section 4.3).
14: $M = \arg\min L(G, M) = \arg\min\{L(M) + L(\mathbf{E}) + L(\mathbf{O})\}$          $\{$Eq. (1),(2),(8)-(10)$\}$
15: // Module D: Approximate supergraph $G_S(\mathcal{V}_S, \mathcal{E}_S)$ creation conditional on the discovered patterns
16: // (supernodes linked via weighted superedges).
17: $\mathcal{V}_S = \{s \in M\}$                                   $\{$supernodes = structures in $M\}$
18: $\mathcal{E}_S = \{(s_i, s_j, w_{ij}) \mid w_{ij} = |\{u, v\}|$, node $u \in s_i$, node $v \in s_j, i \neq j\}$          $\{$superedges$\}$
19: return approximate supergraph $G_S(\mathcal{V}_S, \mathcal{E}_S)$ (summary $M$)
```

not account for overlaps, which is based on the modeling described in Sections 3.1 and 3.2, includes all three structures in the summary, yielding both redundant nodes and edges. Despite the overlap, the description length of the graph given the model above is calculated as **441** *bits, since edges that are covered multiple times are not penalized. For reference, the graph needs* **652** *bits under the null (empty) model, where all original edges are captured in the error matrix* $\mathbf{E}$. *Ideally, we want a method that penalizes extensive overlaps and maximizes node/edge coverage.*

In the example that we described above, by leveraging the full optimization function in Equation (1)*, which includes the edge-overlap penalty term, we obtain a summary with only the first two cliques, as desired. The encoding of our proposed method has a length of* **518** *bits, which is higher than the number of bits of the non edge-overlap aware encoding (441 bits). The reason is that in the former (edge-overlap aware) summary, some edges have remained unexplained (edges from nodes 11-20 to nodes 21-40), and thus are encoded as error. On the other hand, the latter summary encodes all the nodes and edges without errors, but explains many edges twice (e.g., the nodes 11-20 and the edges between them, the edges between nodes 11-20 and 21-30) without accounting for the redundancy-related bits.*

Our proposed edge-overlap aware encoding can effectively handle a family model $\mathcal{M}$ that consists of subsets of node- and edge-overlapping structural patterns. This encoding chooses a model M that describes the input graph well while minimizing redundant modeling of nodes and edges.

4 CONDENSE: Our Proposed Algorithm

Based on the model from Section 3, we propose CONDENSE, an ensemble, edge-overlap-aware algorithm that summarizes a graph with a compact supergraph consisting of a diverse set of structural patterns (e.g., *fc*, *hs*). CONDENSE consists of four modules (Algorithm 1), described in detail next.

4.1 Module A: Unified Pattern Discovery Module

As we mentioned earlier, in our formulation, we consider summaries in the model family $\mathcal{M}$, which consists of all possible permutations of subsets of structural patterns in Ω (e.g., a summary with 10 full cliques, 3 bipartite cores, 5 stars and 9 hyperbolic structures). Towards this goal, the first step is to discover subgraphs in the input graph. These can then be used to build its summary. To find the "perfect" graph summary, we would need to generate all possible (2^n) patterns for a given graph G, and then, from all possible (2^{2^n}) combinations of these patterns pick the set that minimizes Equation (1). This is intractable even for small graphs. For example, for $n = 100$ nodes, there are more than $2^{\text{nonillion}}$ (1 nonillion $= 10^{30}$) possible summaries. We reduce the search space by considering patterns that are found via graph clustering methods, and are likely to fit the structural patterns in Ω well.

The literature is rich in graph clustering methods [8, 25, 26, 49]. However, each approach is biased toward identifying specific types of graph structures, which are most often cliques and bipartite cores. Choosing a decomposition method to generate patterns for the summary depends on the domain, the expected patterns (e.g., mainly clique- or star-like structures), and runtime constraints. To mitigate biases introduced by individual clustering methods, and to consider a diverse set of candidate patterns, we propose a unified approach that leverages seven existing clustering methods: SLASHBURN, LOUVAIN, SPECTRAL, METIS, HYCOM, BIGCLAM, and KCBC (Section 2). In Table 1, we present the qualitative advantages, disadvantages, and biases of the methods. Specifically, SLASHBURN tends to provide excellent graph coverage and biased summaries in which stars dominate. Conversely, most other approaches produce primarily full cliques and stars, and some bipartite cores. HYCOM finds mainly hyperbolic communities with skewed degree distributions.

Our proposed unified approach (Algorithm 1, lines 2-4) is expected to lead to summaries with a better balanced set of structures (i.e., a good mix of exact and approximate cliques, bipartite cores, stars, chains and hyperbolic structures), and lower encoding cost than any standalone graph clustering method. At the same time, it is expected to take longer to generate all the patterns (although the clustering methods can trivially run in parallel) and the search space for the summary becomes larger, equal to the union of all the subgraphs that the clustering methods generate.

In the experimental evaluation, we use CONDENSE to empirically compare the impact of these methods on the summary quality and evaluate their summarization power.

4.2 Module B: Structural Pattern Identification Module

This module (Algorithm 1, lines 5-12) identifies and assigns an identifier structural pattern in Ω to all the subgraphs found in module A. In other words, this module seeks to characterize each cluster with its best-suited pattern in $\Omega = \{fc, st, bc, ch, hs\}$. Let g be the induced graph of a pattern generated in Step 1, and ω be a pattern in Ω. Following the reasoning in Section 3, we use MDL as a selection criterion. To model g with ω, we first model g with its best representation as structure type ω (explained in detail next), $r(g, \omega)$, and define its encoding cost as $L_{r(g,\omega}(g, \omega) = L(\omega) + L(g|\omega) = L(\omega) + L(E_\omega^+) + L(E_\omega^-)$, where E_ω^+ and E_ω^- encode the erroneously modeled and unmodeled edges of g. The pattern type in ω that leads to the smallest MDL cost is used as the identifier of the corresponding subgraph g (lines 11-12 in Alg. 1).

Finding the best representation $r(g, \omega)$. Per pattern type ω, each pattern g can be represented by a family of structures—e.g., we can represent g with as many bipartite cores as can be induced on all possible permutations of g's nodes into two sets L (left nodeset) and R (right nodeset) . The only exception is the full clique (*fc*) pattern, which has a unique (unordered) set of nodes. To make the problem tractable, we use the graph-theoretical properties of the pattern types in Ω in order to choose the representation of g which minimizes the incorrectly modeled edges.

Specifically, we represent g as a star by identifying its highest-degree node as the hub and all other nodes as spokes. Representing g as a bipartite core reduces to finding the maximum bipartite pattern, which is NP-hard. To scale-up the computation, we approximate it with semi-supervised classification with two classes L and R, and the prior information that the highest-degree node belongs to L and its neighbors to R. For the classification, we use Fast Belief Propagation [30] with heterophily between neighbors. Similarly, representing g as a chain reduces to finding its longest path, which is also NP-hard. By starting from a random node, we perform breadth-first search two times, and end on nodes v_1 and v_2, respectively. Then, we consider the path v_1 to v_2 (based on BFS), and perform local search to further expand it. For the hyperbolic structures, we used power-law fitting (`http://tuvalu.santafe.edu/~aaronc/powerlaws/` by Clauset et al.). Lines 7-10 in Algorithm 1 succinctly describe the search of the best representation r for every subgraph g and pattern type ω.

4.3 Module C: Structural Pattern Selection Module

This module is key for creating *compact* summaries and is described in lines 13-14 of Alg. 1. Ideally, we would consider all possible combinations of the previously identified structures and pick the subset that minimizes the encoding cost in Equation (1). If $|\mathcal{S}|$ structures have been found and identified in the previous steps, finding the optimal summary from $2^{|\mathcal{S}|}$ possibilities is not tractable. For reference, we have seen empirically that graphs with about 100,000 nodes, have over $50K$ structures. The optimization function is neither monotonic nor submodular, in which case a greedy hill climbing approach would give a $(1-\frac{1}{\epsilon})$-approximation of the optimal.

Instead of considering all possible combinations of structures for the summary, prior work has proposed GNF, a heuristic that considers the structures in decreasing order of "local" encoding benefit and includes in the model the ones that help further decrease the graph's encoding cost $L(G,M)$. The local encoding benefit [29] is defined as $L(g,\emptyset)-L(g,\omega)$, where $L(g,\emptyset)$ represents the encoding of g as noise (i.e., empty model). Although it is efficient, its output summary and performance heavily depend on the structure order. To overcome these shortcomings and obtain more compact summaries, we propose a new structural pattern selection method, STEP, as well as a faster serial version and three parallel variants: STEP-P, STEP-PA, and K-STEP.

- **STEP.** This method iteratively sifts through all the structures in $\mathcal{S}$ and includes in the summary the structure that decreases the cost in Equation (1) the most, until no structure further decreases the cost. Formally, if $\mathcal{S}_i$ is the set of structures that have not been included in the summary at iteration i, STEP chooses structure s_i^* s.t.
$$s_i^* = arg \min_{s\in\mathcal{S}_i} L(G, M_{i-1}\cup\{s\})$$
where M_{i-1} is the model at iteration $i-1$, and $M_0=\emptyset$ is the empty model. CONDENSE with STEP finds up to 30% more compact summaries than baseline methods, but its quadratic runtime $O(|\mathcal{S}|^2)$ makes it less ideal for large datasets with many structures $\mathcal{S}$ produced by module A. Therefore, we propose four methods that significantly reduce STEP's runtime while maintaining its summary quality.

- **STEP-P.** The goal of STEP-P is to speed up the computation of STEP by iteratively solving smaller, “local” versions of STEP in parallel. STEP-P begins by dividing the nodes of the graph into p partitions using METIS. Next, each candidate structural pattern is assigned to the partition with the maximal node overlap. STEP-P then iterates until convergence, with each iteration consisting of two phases:
  1. **Parallelize.** In parallel, a process is spawned for each partition and is tasked with finding the structure that would lower the encoding cost in Eq. (1) the most out of all the structures in its partition. For any given partition, there may be no structure that lowers the global encoding cost.
  2. **Sync.** From all structures returned in phase 1, the one that minimizes Equation (1) the most is added to the summary. If no structure reduces the encoding cost, the algorithm has converged. If not, phase 1 is repeated.
- **STEP-PA.** In addition to parallelizing STEP, we introduce the idea of “inactive” partitions, which is an optimization designed to reduce the number of processes that are spawned by STEP-P. STEP-PA differs from STEP-P by designating every partition of the graph as active, then if a partition fails x times to find a structure that lowers the cost in Equation (1), that partition is declared inactive and is not visited in future iterations. Thus, the partitions with structures not likely to decrease the overall encoding cost of the model get x chances (e.g. 3) before being eventually ruled out, effectively reducing the number of processes spawned for each iteration of STEP-PA after the first x iterations.
- **K-STEP.** The pseudocode of this variant is given in Algorithm 2. K-STEP further speeds up STEP while maintaining high-quality summaries. This algorithm has two phases: the first applies STEP-P K times (lines 3-5) to guarantee that the initial structural patterns included in the summary are of good quality. The second expands the summary by building local solutions of STEP-P per active partition (lines 8-9). If a partition does not return any solution, it is flagged as inactive (lines 10-11). For the partitions that returned non-empty solutions, the best structure per partition is added into a temporary list (line 13), and a parallel “glocal” step applies STEP-P over that list and populates the summary (lines 14-16). We refer to this step as “glocal” because it is a global step within the local stage. The local stage is repeated until no active partitions are left.

4.4 Module D: Approximate Supergraph Creation Module

In the empirical analysis (Section 5), we show that STEP results in graph summaries with up to 80-90% fewer structures than the baselines, and thus can be leveraged for tractable graph visualization. The last and fourth module of CONDENSE (Algorithm 1, lines 15-18), instead of merely outputting a list of structures, creates an “approximate” supergraph which gives a high-level but informative view of large graphs. An *exact* supergraph, $G_S(\mathcal{V}_S, \mathcal{E}_S)$, of a graph $G(\mathcal{V}, \mathcal{E})$ consists of a set of supernodes $\mathcal{V}_S = P(\mathcal{V})$ which is a power set (i.e., family of sets) over $\mathcal{V}$ and a set of superedges $\mathcal{E}_S$. The superweight is often defined as the sum of edge weights between the supernodes’ constituent nodes.

Unlike most prior work, CONDENSE creates “approximate,” yet powerful supergraphs: (i) the supernodes do not necessarily correspond to a set of nodes with the same connectivity, but to *rich* structural patterns (including hyperbolic structures and chains); (ii) the supernodes may have *node overlap*, which helps to pinpoint bridge nodes (i.e., nodes that span multiple communities); (iii) the supernodes may show deviations from the perfect corresponding structural patterns (i.e., they correspond to near-structures).

Algorithm 2 K-STEP

```
1: Input: graph $G(\mathcal{V},\mathcal{E})$; list of structures $\mathcal{S}$; $P$ partitions; K iterations
2: ActivePartitions = $\{1,\ldots,P\}$                      {all partitions are active}
3: // Stage 1: Global
4: for $i = 1 : \text{K}$
5:     run STEP-P ()                                        {summary of K structures}
6: // Stage 2: Local Stage
7: repeat:
8:     for $p \in$ ActivePartitions:                        {2.1: Local sub-stage }
9:         $s$ = run STEP-P-Parallelize()                    {s = best structure in $p$}
10:        if $s = \emptyset$                                {no structure returned}
11:            ActivePartitions.remove(p)                    {partition $p$ is inactive}
12:        else
13:            bestStructs.add(s)                            {$s$ is candidate for $M$}
                                                             {$p$ remains active}
14:    repeat:                                   {in parallel, add structures to $M$}
15:        run STEP-P-Sync(bestStructs)                      {2.2: Glocal sub-stage }
16:    until bestStructs = $\emptyset$ or Eq. (1) is minimized
17: until ActivePartitions $= \emptyset$
18: return $M$
```

DEFINITION. A **CONDENSE approximate supergraph** of G is a supergraph with supernodes that correspond to *possibly-overlapping structural patterns* in Ω. These patterns are approximations of clusters in G.

In other words, the CONDENSE supergraphs consist of supernodes that are *fc*, *st*, *ch*, *bc*, and *hs*. To obtain an approximate supergraph, we map the structural patterns returned in module C to approximate supernodes. Then, for every pair of supernodes, we add a superedge if there were edges between their constituent nodes in $\mathcal{V}$ and set its superweight equal to the number of such (unweighted) edges, as shown in line 18 of Algorithm 1.

To evaluate the edge overlap in the summaries, and hence the effectiveness of our overlap-aware encoding, we use the normalized overlap metric, which between two supernodes is equivalent to the Jaccard similarity. This value is 0 if the supernodes do not share any nodes, and close to 1 if they share many nodes compared to their sizes. Although we do not focus on these tasks here, the CONDENSE supergraphs can be used for visualization and potentially for approximation of algorithms on large networks (without specific theoretical guarantees, at least in the general form).

4.5 CONDENSE: Complexity Analysis

In the complexity analysis of CONDENSE, we consider each module separately: The first module has complexity $O(n^3)$, which corresponds to SPECTRAL. However, in practice, HYCOM is often slower than SPECTRAL, likely due to implementation differences (JAVA vs. MATLAB). The complexity of this module can be lowered by selecting the fastest methods. Module B is linear on the number of edges of the discovered patterns. Given that they are overlapping, the computation of $L(G,M)$ is done in $T = O(|M|^2 + m)$, which is $O(m)$ for real graphs with $|M|^2 << m$. In module C, STEP has complexity $O(|S|^2 \times T)$, where S is the set of labeled structures. STEP-P and STEP-PA are $O(t \times \frac{|S|^2}{p} \times T)$, where p is the number of METIS partitions ('active' partitions for STEP-PA) and t is the number of iterations. K-STEP is a combination of STEP-P and a local stage, so it runs in $O(K \times \frac{|S|^2}{p} \times T + t_{lcl} \times (\frac{|S|^2}{p_{active}} + p_{active}^2) \times T)$, where t_{lcl} is the iterations of its local stage. Finally, the supergraph (module D) can be generated in $O(m)$.

5 Empirical Analysis

We conduct thorough experimental analysis to answer three main questions:

- How effective is CONDENSE?
- Does it scale with the size of the input graph?
- How do the clustering methods compare in terms of summarization power?

Setup. We ran experiments on the real graphs given in Table 3. With regards to clustering parameters, we choose the number of SLASHBURN hub nodes to slash per iteration $h_{slash} = 2$ to achieve better granularity of clusters. For LOUVAIN, we choose resolution $\tau = 0.0001$ as it generates a number of clusters comparable with other clustering methods for all our datasets. For SPECTRAL and METIS, the number of clusters k are set to $\sqrt{n/2}$ according to a rule of thumb [1], where n is the number of nodes in the graph. The other clustering methods are parameter-free. Unless otherwise specified, we followed the same rule of thumb for setting the number of input METIS partitions p for all the STEP variants. In subsections 5.1 and 5.2, we set the number of chances $x = 3$ for STEP-PA.

Table 3: Summary of graphs used in our experiments.

Name	Nodes	Edges	Description
`EUmail` [34]	265,214	420,045	EU university email communications
`Enron` [34]	80,163	288,364	Enron email communications
`AS-Caida` [34]	26,475	106,762	BGP routing table
`AS-Oregon` [34]	13,579	37,448	Router connections
`Choc`	2,899	5,467	Co-editor Wikipedia graph

5.1 Effectiveness of CONDENSE

Ideally, we want a summary to be: (i) concise, with a small number of structures/supernodes; (ii) minimally redundant, i.e., capturing dependencies such as *overlapping* supernodes, but without overly encoding overlaps; and (iii) covering in terms of nodes and edges. We perform experiments on the real data in Table 3 to evaluate CONDENSE's performance with regard to these criteria. We note that we do not evaluate the effectiveness of CONDENSE by comparing structural properties of the compressed graph to those of the original graph, since the goal of our method is to detect 'important' structures (which can help with better understanding the underlying patterns) and does not optimize for specific graph properties or queries (that is usually the goal of graph sampling techniques).

Baselines. The first baseline is VOG [29], which we describe in Section 2. For our experiments, we used the code that is online at `https://github.com/GemsLab/VoG_Graph_Summarization`. The second baseline is our proposed method, CONDENSE, combined with the GNF heuristic (Section 4.3).

A1. Conciseness. In Table 4, we compare our proposed method (for different structure selection methods) and the baselines with respect to their compression rates, i.e., the percentage of bits needed to encode a graph with the composed summary over the number of bits needed to encode the corresponding graph with an empty model/summary (that is, all its edges are in the error matrix). In parentheses, we also give the total number of structures in the summaries.

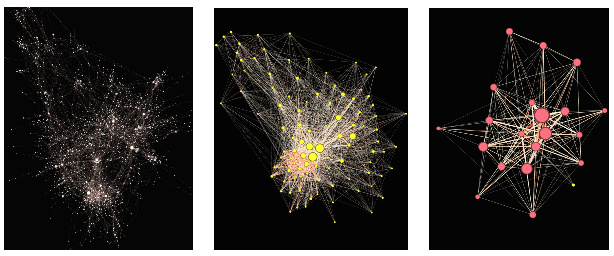


Fig. 3: `Choc`: CondeNSe-Step generates more compact supergraphs. (b)-(c): The full supergraphs by VoG-GnF, and CondeNSe-Step, resp. Yellow for stars, red for cliques, green for bipartite cores. The edge weights correspond to the number of inter-supernode edges.

We find that compared to baselines, CondeNSe with the Step variants yields significantly more compact summaries, with 30%–50% lower compression rate and about 80–90% fewer structures. The Step variants give comparable results in summarization power.

Table 4: CondeNSe: Compression rate with respect to the empty model (i.e., percentage of bits used with the model vs. the empty model). In parentheses, we give the number of structures in the corresponding summary. A "-" means that the corresponding method was terminated after 4 days. (* In the interest of time, the summary size was limited to 15.)

Dataset	**VoG [29]**	**CondeNSe** GnF	**CondeNSe with Step Variants**			
			Step	**Step-P**	**Step-PA**	**k-Step**
`Choc`	88% (101)	88% (101)	56% (24)	56% (24)	56% (21)	56% (22)
`AS-Oregon`	71% (400)	69% (379)	35% (41)	35% (41)	35% (35)	35% (36)
`AS-Caida`	-	71% (572)	42% (51)	42% (51)	42% (46)	44% (60)
`Enron`	75% (2330)	74% (2044)	-	26% (50)	25% (201)	25% (218)
`EUmail`	-	65% (1440)	-	-	-	59% (15*)

A2. Minimal Redundancy. In Figures 1 and 3, we visualize the supergraphs for `AS-Oregon` and `Choc`, which are generated from the selected structures of VoG and CondeNSe-Step. It is visually evident that the CondeNSe supergraphs are significantly more compact. In Table 5, we also provide information about the number of overlapping supernode pairs and their average Jaccard similarity, as an overlap quantifier (in parentheses). For brevity, we only give results for k-Step, since the results of the rest Step-series are similar. We observe that CondeNSe has significantly fewer supernode overlaps, and the overlaps are smaller in magnitude. We also note that the overlap encoding module achieves 10-20% reduction in overlapping edges, showing its effectiveness for minimizing redundancy.

Table 5: Overlapping supernode pairs and average similarity in parentheses. CondeNSe reduces the overlap. A "-" means that the corresponding method was terminated after 4 days.

Dataset	**VoG [29]**	**CondeNSe**
`Choc`	900 (0.04)	74 (0.029)
`AS-Oregon`	15875 (0.047)	126 (0.026)
`AS-Caida`	-	382 (0.018)
`Enron`	447052 (0.02)	509 (0.015)
`EUmail`	-	0

A3. Coverage. We give the summary node/edge coverage as a ratio of the original for different assembly methods in Figure 4. We observe that the baselines have better edge coverage than the Step variants, which is expected as they include significantly more structures in their summaries. However, in most cases, k-Step and Step-PA achieve better node coverage than the baselines. Taking into account the (contradicting) desired property for summary

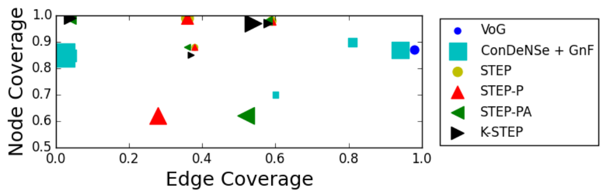


Fig. 4: Node coverage vs. edge coverage—marker size corresponds to the graph size. STEP variants have better node coverage, and handle the summary coverage-conciseness trade-off well.

Table 6: CONDENSE: Number of structures per type in the summaries in the format $[fc, st, bc]$, for VoG we have $[fc + nc, st, bc + nb]$, where nc is near-clique and nb is near-bipartite core. The CONDENSE summaries are more balanced, *without* a specific pattern type dominating in all the graphs. In the interest of time, we find the top-50 and top-15 structures for `Enron` and `EUmail`, respectively. A "-" means that the corresponding method was terminated after 4 days.

Dataset	**VoG [29]**	**CONDENSE-GNF**	**CONDENSE with STEP Variants**			
			STEP	**STEP-P**	**STEP-PA**	**K-STEP**
`Choc`	[0,101,0]	[1,100,0]	[21,3,0]	[21,3,0]	[20,1,0]	[21,1,0]
`AS-Oregon`	[1,399,0,]	[19,355,5]	[27,13,1]	[27,13,1]	[26,9,0]	[26,10,0]
`AS-Caida`	-	[2,557,13]	[38,7,6]	[38,7,6]	[37,5,4]	[43,12,5]
`Enron`	[2,2323,5]	[160,1676,208]	-	[45,2,3]	[60,108,33]	[61,124,33]
`EUmail`	-	[0,1261,179]	-	-	-	[15,0,0]

conciseness, CONDENSE with STEP variants has better performance, balancing coverage and summary size well.

What other properties do the various summaries have? What are the main structures found in different types of networks (e.g., email vs. routing networks)? In Table 6, we show the number of in-summary structures per type. We note that no chains and hyperbolic structures were included in the summaries of the networks that we show here (although some were found by the pattern discovery module, and there are synthetic examples in which they are included in the final summaries). This is possibly because stars are extreme cases of hyperbolic structures, and the encoding of (approximate) hyperbolic structures is of the same order, yet often more expensive than the encoding of stars with errors. As for chains, they are not 'typical' clusters found by popular clustering methods, but rather by-products of the decomposition methods that we consider. Moreover, given that the chain encoding considers the sequence of node IDs, and errors in the real data increase the encoding cost, very often encoding them in the error matrix yields better compression. One observation is that STEP gives less biased summaries than the baselines. For email networks, we see that stars are dominant (e.g., users emailing multiple employees that do not contact each other), with considerable number of cliques and bipartite cores too. For routing networks (`AS-Caida` and `AS-Oregon`), we mostly see cliques (e.g., "hot-potato" routing), and a few stars and bipartite cores. In collaboration networks, cliques are the most common structures, followed by stars (e.g., administrators). VoG and CONDENSE-GNF are biased towards stars, which exceed the other structures by an order of magnitude. Overall, CONDENSE fares well with respect to the desired properties for graph summaries.

5.2 Runtime Analysis of CONDENSE

We give the runtime of pattern discovery and the STEP methods in Figure 5. As we discussed in the complexity analysis, the modules of our summarization method depend on the number of edges and selected structures. Thus, in the figure we plot the runtime of our variants with respect to the number of edges in the input graphs."Discovery" represents the maximum time of the clustering methods, and "Disc.-Fast" corresponds to the slowest among the fastest methods (KCBC, LOUVAIN, METIS, BIGCLAM). We ran the experiment on an Intel(R) Xeon(R) CPU E5-1650 at 3.50GHz, and 256GB memory.

We see that the fast unified discovery is up to $80\times$ faster than the original one. As expected, STEP is the slowest method. The parallel variants STEP-P, STEP-PA, and K-STEP are more scalable, with K-STEP being the most efficient. Taking into account the similarity of the heuristics in both conciseness and coverage, Figure 5 further suggests that K-STEP is the best-performing heuristic given that it exhibits the shortest runtime.

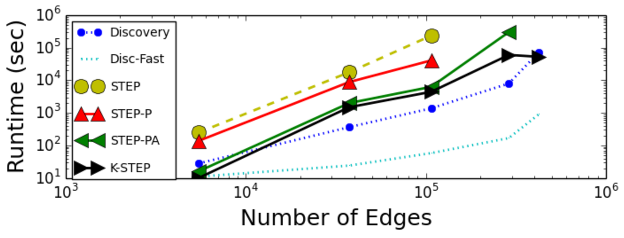


Fig. 5: Runtime vs. # of edges: K-STEP is more efficient than the other methods, and scales to larger graphs.

5.3 Sensitivity Analysis of CONDENSE: Agreement between STEP and STEP-variants

Our analysis so far has shown that K-STEP leads to the best combination of high compression and low runtime compared to the other methods. But how well does it approximate STEP in terms of the generated summary? To answer this question, we evaluate the "agreement" between the generated summaries, which in this section we view as ordered lists of structures based on the iteration they were included in the final summary (which defines the rank of each structure). Many measures have been proposed to quantify the correlation between sequences, including the famous Pearson's product-moment coefficient. And when it comes to rank correlation measures, Spearman's ρ and Kendall's τ are the popular ones, while others are mostly ad hoc and not pervasive. These measures, however, only work on permuted lists or lists of the same length, while the generated summaries can have different constituent structures and lengths. Other measures that are popular in information retrieval, such as precision@k, cannot explain in detail the level of disagreement between two models (i.e., ranked lists) as they treat each summary as an unordered set of structures. In our evaluation, we want a measure that (i) effectively handles summaries of different lengths, and (ii) penalizes with different, adaptive weights 'rank' disagreement between structures included in both summaries, and disagreement for missing structures from one summary. Thus, we propose a new measure of agreement between models, which we call AG. Let M_1 and M_2 be the two summaries, and $rank(s, M_i)$ be the ranking of structure s in summary M_i (i.e., the order in which it was included in the summary while minimizing Eq. (1)). We define the agreement of the two summaries as:

$$AG(M_1, M_2) = 1 - \text{normalized disagreement} = 1 - \frac{1}{Z}[\alpha D + (1-\frac{\alpha}{2})D_1 + (1-\frac{\alpha}{2})D_2]$$

where the disagreement has three components: (i) $D = \sum_{s\in M_1\cap M_2} |rank(s, M_1) - rank(s, M_2)|$ is the rank disagreement for structures that are in both summaries, (ii) $D_1 = \sum_{s\in M_1\cap M_2'} |(|M_2|+$

Table 7: Agreement of STEP and its variants. They approximate STEP quite well. (* Agreement based on the top-50 structures due to STEP-P's lack of scalability.)

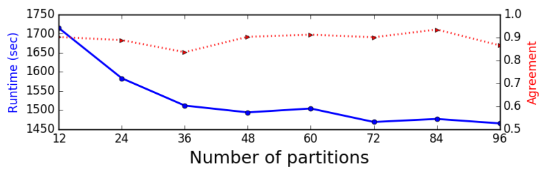
Dataset	STEP-P	STEP-PA	K-STEP
`Choc`	1	0.9886	0.9667
`AS-Oregon`	1	0.9704	0.9285
`AS-Caida`	1	0.9865	0.8238
`Enron`	1	0.5012*	0.446*

$1) - rank(s, M_1)|$ is the disagreement for structures in M_1 but not in M_2, and (iii) D_2 is defined analogously to capture the disagreement for structures in M_2 but not in M_1. Finally, Z is a normalization factor that guarantees that the normalized disagreement, and thus AG, are in $[0, 1]$: $Z = (1-\frac{\alpha}{2})\sum_{s\in M_1} |(|M_2|+1) - rank(s, M_1)| + \sum_{s\in M_2} |(|M_1|+1) - rank(s, M_2)|$. Based on the definition above, $AG = 1$ for identical summaries, while 0 for completely different summaries. In order to penalize more the structures that appear in one summary but not in the other, we set $\alpha = 0.3$ (the results are consistent for other values of α). In Table 7, we give the agreement between STEP and its faster variants. As a side note, the agreement with VOG is almost 0 in all the cases. As expected, STEP-P produces the same summaries as STEP, while STEP-PA and K-STEP preserve the agreement well.

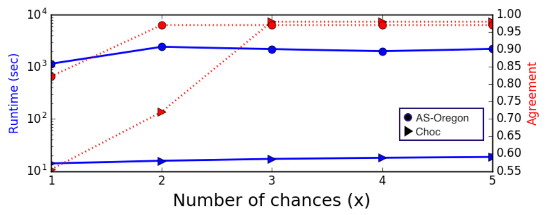
5.4 Sensitivity to the number of partitions

All parallel variants of STEP take p METIS partitions as input. To analyze the effects of varying p on runtime and agreement, we ran K-STEP and increased p from 12 to 96 in increments of 12.

We only give the results on `AS-Oregon`, since other datasets lead to similar results. We observe that while agreement is robust, runtime decreases as p increases and especially so with the smaller values of p. This observation is consistent with our motivation for parallelizing STEP: by decreasing the number of structures in any given partition, the "local" subproblems of STEP become smaller and thus less time-consuming. Figure 6a shows the effect of the number of partitions on runtime and agreement, both averaged over three trials.



(a) `AS-Oregon`: Runtime and agreement vs. number of partitions.



(b) `Choc` + `AS-Oregon`: Runtime and agreement vs. number of chances (x).

Fig. 6: The agreement is robust to the number of partitions, while the runtime decreases.

5.5 Sensitivity of STEP-PA

We also experimented with varying the number of “chances” allowed for partitions in the STEP-PA variant. STEP-PA speeds up STEP-P by forcing partitions to drop out after not returning structures for a certain number of attempts (x). However, while giving partitions fewer chances can speed up the algorithm, smaller values of x can compromise compression and agreement.

In Figure 6b, we give the agreement and runtime of STEP-PA on `Choc` and `AS-Oregon` setting $x = \{1, 2, 3, 4, 5\}$. We found that both runtime and agreement increased with x, and plateaued after $x = 3$. This suggests that forcing partitions to drop out early, while better for runtime, can lead to the loss of candidate structures that may be useful for compression later.

5.6 CONDENSE as a Clustering Evaluation Metric

Given the independence of STEP from the structure ordering, we use CONDENSE to evaluate the different clustering methods and give their individual compression rates in Table 8.

For number and type of structures we give our observations based on `AS-Oregon` (Figure 7), which is consistent with other datasets. As we see in the case of `AS-Oregon` (which is consistent with the other networks), SLASHBURN mainly finds stars (136 out of 138 structures); LOUVAIN, SPECTRAL, KCBC, and BIGCLAM reveal mostly cliques (9/9, 15/17, 9/9, and 28/29, respectively); METIS has a less biased distribution (18 cliques, 12 stars), and HYCOM, though looks for hyperbolic structures, tends to find cliques in our experiments (45 out of 52 structures). Also, SLASHBURN and BIGCLAM discover more structural patterns than other methods, which partially explains their good compression rate in Table 8.

We perform an ablation study to evaluate the graph clustering methods in the context of summarization. Specifically, we create a leave-one-out unified model for each clustering method and evaluate the contribution of each clustering method to the final summary. The results are shown in Table 9. We see that LOUVAIN appears to be the most important method: when included, it contributes the most; and when dropped, the compression rate reduces

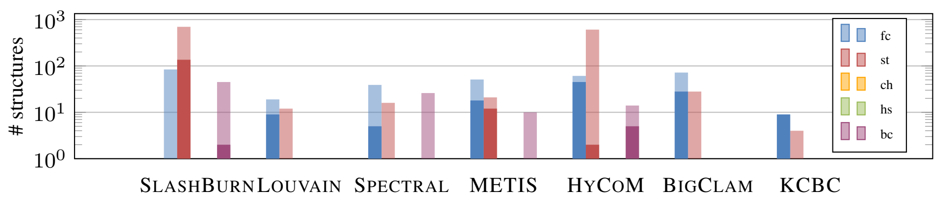


Fig. 7: Number of structures found by the clustering methods for `AS-Oregon`. Transparent/solid rectangles for before/after the structure selection step. Notation: fc: full clique, st: star, ch: chain, bc: bipartite core, hs: hyperbolic structure.

Table 8: CONDENSE as an evaluation metric: Compression rate of clustering methods with respect to the empty model (i.e., percentage of bits for encoding the graph given the chosen model vs. the empty model).

Dataset	Clustering Methods						
	SLASHBURN	LOUVAIN	SPECTRAL	METIS	HYCOM	BIGCLAM	KCBC
`Choc`	88%	99%	99%	100%	100%	87%	**78%**
`AS-Oregon`	76%	94%	82%	85%	98%	83%	**65%**
`AS-Caida`	**70%**	100%	100%	98%	98%	91%	74%

Table 9: Ablation study for AS-Oregon. LOUVAIN and SLASHBURN contribute most to the CONDENSE summaries.

Clustering Method	Compression Rate	Contribution per Method						
		SLASHBURN	LOUVAIN	SPECTRAL	METIS	HYCOM	BIGCLAM	KCBC
SLASHBURN	22%	-	**63%**	10%	7%	7%	0	13%
LOUVAIN	**30%**	30%	-	16%	**45%**	0	3%	7%
SPECTRAL	22%	32%	**51%**	-	3%	0	0	14%
METIS	22%	34%	**46%**	5%	-	2%	0	12%
HYCOM	22%	35%	**48%**	3%	3%	-	0	13%
BIGCLAM	22%	34%	**46%**	2%	2%	2%	-	12%
KCBC	25%	**50%**	35%	6%	2%	2%	6%	-

(worse). When KCBC is dropped, SLASHBURN gets to the top, but LOUVAIN also has considerable contribution. In the missing-LOUVAIN case, the contribution gets redistributed among other clustering methods to make up for it, this effect differs by dataset, e.g., METIS gets boosted for AS-Oregon, while it is SPECTRAL for Choc.

In terms of runtime, for modules A and B (pattern discovery and identification), SPECTRAL and HYCOM take the longest time, while KCBC, LOUVAIN, METIS, and BIGCLAM are the fastest ones, with SLASHBURN falling in the middle. For Module C (summary assembly), the trade-off between runtime and candidate structures is given in the complexity analysis (Appendix 4.5). In practice, HYCOM usually takes the longest time, followed by SPECTRAL and SLASHBURN.

6 Conclusion

In this work we proposed CONDENSE, a method that summarizes large graphs as small, approximate and high-quality supergraphs conditioned on diverse pattern types. CONDENSE features a new selection method, STEP, which generates summaries with high compression and node coverage. However, this comes at the cost of increased runtime, which we addressed by introducing faster parallel approximations to STEP. We provided a thorough empirical analysis of CONDENSE, and contributed a novel evaluation of clustering methods in terms of summarization power, complementing the literature that focuses on classic quality measures. We showed that each clustering approach has its strengths and weaknesses and make different contributions to the final summary. Moreover, CONDENSE leverages their strengths, handles edge-overlapping structures, and shows results superior to baselines, including significant improvement in the bias of summaries with respect to the considered pattern types.

Ideally without the constraint of time, we naturally recommend the application of as many clustering methods in Module A of CONDENSE. On the other hand, to deal with the additional complexity of having more structures, we recommend choosing faster clustering methods or a mixture of fast and 'useful' methods (depending on the application at hand) that contribute good structures, as shown in our analysis.

Acknowledgements

This material is based upon work supported the University of Michigan. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding parties.

5555videcommand[1]

References

1. clusterMaker: Creating and Visualizing Cytoscape Clusters. `http://www.cgl.ucsf.edu/cytoscape/cluster/clusterMaker.shtml`.
2. A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd International Conference on World Wide Web (WWW), Rio de Janeiro, Brazil*. International World Wide Web Conferences Steering Committee, 2013.
3. A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2), 1972.
4. M. Araujo, S. Günnemann, G. Mateos, and C. Faloutsos. Beyond blocks: Hyperbolic community detection. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Nancy, France*, 2014.
5. L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *Proceedings of the 12th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Philadelphia, PA*, 2006.
6. L. Backstrom, R. Kumar, C. Marlow, J. Novak, and A. Tomkins. Preferential behavior in online groups. In *Proceeding of the 1st ACM International Conference on Web Search and Data Mining (WSDM)*, 2008.
7. J. D. Batson, D. A. Spielman, N. Srivastava, and S. Teng. Spectral sparsification of graphs: theory and algorithms. *Commun. ACM*, 56(8), 2013.
8. V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast Unfolding of Communities in Large Networks. *JSTAT*, 2008(10), 2008.
9. D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Seattle, WA*, 2004.
10. F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On Compressing Social Networks. In *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, 2009.
11. R. Cilibrasi and P. Vitányi. Clustering by Compression. *IEEE Transactions on Information Technology*, 51(4), 2005.
12. D. J. Cook and L. B. Holder. Substructure Discovery Using Minimum Description Length and Background Knowledge. *Journal of Artificial Intelligence Research*, 1, 1994.
13. T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.
14. C. Faloutsos and V. Megalooikonomou. On Data Mining, Compression and Kolmogorov Complexity. In *Data Mining and Knowledge Discovery*, volume 15. Springer-Verlag, 2007.
15. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On Power-law Relationships of the Internet Topology. *Proceedings of the ACM SIGCOMM 1999 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Cambridge, MA*, 1999.
16. S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3), 2010.
17. C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. Evaluating cooperation in communities with the k-core structure. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2011.
18. M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *PNAS*, 99, 2002.
19. O. Goonetilleke, D. Koutra, T. Sellis, and K. Liao. Edge labeling schemes for graph data. In *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management (SSDBM), Chicago, IL*, pages 12:1–12:12. ACM, 2017.
20. J. P. Hespanha. An efficient matlab algorithm for graph partitioning. *Department of Electrical and Computer Engineering, University of California, Santa Barbara*, 2004.
21. C. Hübler, H.-P. Kriegel, K. Borgwardt, and Z. Ghahramani. Metropolis Algorithms for Representative Subgraph Sampling. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM), Pisa, Italy*, 2008.
22. D. Jin and D. Koutra. Exploratory analysis of graph data by leveraging domain knowledge. In *Proceedings of the 17th IEEE International Conference on Data Mining (ICDM), New Orleans, LA*, pages 187–196, 2017.
23. D. Jin, A. Leventidis, H. Shen, R. Zhang, J. Wu, and D. Koutra. PERSEUS-HUB: Interactive and Collective Exploration of Large-scale Graphs. *Informatics (Special Issue "Scalable Interactive Visualization")*, 4(3), 2017.
24. L. Jin and D. Koutra. Ecoviz: Comparative vizualization of time-evolving network summaries. In *ACM Knowledge Discovery and Data Mining (KDD) 2017 Workshop on Interactive Data Exploration and Analytics*, 2017.

25. U. Kang and C. Faloutsos. Beyond ‘Caveman Communities’: Hubs and Spokes for Graph Compression and Mining. In *Proceedings of the 11th IEEE International Conference on Data Mining (ICDM), Vancouver, Canada*, 2011.
26. G. Karypis and V. Kumar. Multilevel k-way Hypergraph Partitioning. In *Proceedings of the IEEE 36th Conference on Design Automation Conference (DAC), New Orleans, LA*, 1999.
27. J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web as a Graph: Measurements, Models and Methods. In *Proceedings of the International Computing and Combinatorics Conference (COCOON), Tokyo, Japan*, Berlin, Germany, 1999. Springer.
28. D. Koutra and C. Faloutsos. Individual and collective graph mining: Principles, algorithms, and applications. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 9(2):1–206, 2017.
29. D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. VoG: Summarizing and Understanding Large Graphs. In *Proceedings of the 14th SIAM International Conference on Data Mining (SDM), Philadelphia, PA*, 2014.
30. D. Koutra, T.-Y. Ke, U. Kang, D. H. Chau, H.-K. K. Pao, and C. Faloutsos. Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Athens, Greece*, 2011.
31. D. Koutra, V. Koutras, B. A. Prakash, and C. Faloutsos. Patterns amongst Competing Task Frequencies: Super-Linearities, and the Almond-DG Model. In *Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Gold Coast, Australia*, 2013.
32. K. LeFevre and E. Terzi. Grass: Graph structure summarization. In *Proceedings of the 10th SIAM International Conference on Data Mining (SDM), Columbus, OH*. SIAM, 2010.
33. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the 11th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Chicago, IL*. ACM, 2005.
34. J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.
35. J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th International Conference on World Wide Web (WWW), Raleigh, NC*. ACM, 2010.
36. Y. Liu*, T. Safavi*, A. Dighe, and D. Koutra. A graph summarization: A survey. *CoRR*, abs/1612.04883 (To appear in ACM Computing Surveys), 2016.
37. Y. Liu, N. Shah, and D. Koutra. An empirical comparison of the summarization power of graph clustering methods. *Neural Information Processing Systems (NIPS) Networks Workshop, Montreal, Canada*, 2015.
38. A. S. Maiya and T. Y. Berger-Wolf. Sampling Community Structure. In *Proceedings of the 19th International Conference on World Wide Web (WWW), Raleigh, NC*. ACM, 2010.
39. M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Diego, CA*, 2011.
40. S. Navlakha, R. Rastogi, and N. Shrivastava. Graph Summarization with Bounded Error. In *Proceedings of the 2008 ACM International Conference on Management of Data (SIGMOD), Vancouver, BC*, 2008.
41. OCP. Open Connectome Project. http://www.openconnectomeproject.org, 2014.
42. B. A. Prakash, M. Seshadri, A. Sridharan, S. Machiraju, and C. Faloutsos. EigenSpokes: Surprising Patterns and Scalable Community Chipping in Large Graphs. *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Hyderabad, India*, 2010.
43. D. Rafiei and S. Curial. Effectively Visualizing Large Networks Through Sampling. In *IEEE VIS*, 2005.
44. S. Raghavan and H. Garcia-Molina. Representing web graphs. In *Proceedings of the 19th International Conference on Data Engineering (ICDE), Bangalore, India*. IEEE, 2003.
45. J. Rissanen. A Universal Prior for Integers and Estimation by Minimum Description Length. *The Annals of Statistics*, 11(2), 1983.
46. T. Safavi, C. Sripada, and D. Koutra. Scalable hashing-based network discovery. In *Proceedings of the 17th IEEE International Conference on Data Mining (ICDM), New Orleans, LA*, pages 405–414, 2017.
47. N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *Proceedings of the 21st ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Sydney, Australia*. ACM, 2015.
48. D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6), 2011.
49. J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceeding of the 6th ACM International Conference on Web Search and Data Mining (WSDM)*. ACM, 2013.