2024 IEEE International Conference on Knowledge Graph (ICKG) | 979-8-3315-0882-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICKG63256.2024.00009

Substructure Discovery in Heterogeneous Multilayer Networks

Kiran Bolaj, Abhishek Santra and Sharma Chakravarthy
IT Lab and Department of Computer Science & Engineering, The University of Texas at Arlington
kxb0889@mavs.uta.edu, abhishek.santra@mavs.uta.edu, sharmac@cse.uta.edu

***Abstract*—Graph mining analyzes graphs to find core substructures (connected subgraphs) in applications that are modeled using graphs. These identified substructures, based on frequency or some other metric, are important as they reveal an inherent feature or property in the given graph/forest. Interesting substructures that are frequent or compress the graph well offer insights into hidden regularities. The process of finding these interesting patterns using unsupervised learning is termed substructure discovery. SUBDUE is an early main-memory algorithm developed for substructure discovery. Since then, this algorithm has been extended using disk-based, database-oriented approaches, and more recently using the Map/Reduce paradigm to exploit distributed and parallel processing. Graph sizes have increased steadily, thanks to the advent of the Internet and social network applications. To model complex datasets – with multiple types of entities and relationships – multilayer networks (or MLNs) have been shown to be effective. MLNs have also been shown to be superior as compared to simple and attributed graphs for modeling complex data. MLNs are also useful for modeling different data types using distinct layers. This paper focuses on substructure discovery in heterogeneous multilayer networks (one type of MLN) using the novel decoupling-based approach. In this approach, each layer is processed independently and then the results from two or more layers are composed to identify substructures in the entire MLN. The algorithm is designed and implemented, including the composition part, using the Map/Reduce paradigm for understanding speedup. After validating accuracy of results with ground truth, we analyze speedup and response time of the proposed algorithm and approach through extensive experimental analysis on large synthetic datasets with diverse graph characteristics.**

***Index Terms*—Substructure Discovery, Multilayer Networks, Decoupling-approach, Map/Reduce Architecture**

I. Introduction

Large applications that can be modeled using graphs are ubiquitous. Graph models have been used for the analysis of the World Wide Web's structure [1], social-media data, bio-informatics data [2], atoms and covalent relationships in chemistry [3], etc. Graphs are better than other representations for data that embed inherent relationships among objects or entities. Graphs are also easy to understand. Graph models use vertices and edges where each vertex of the graph will correspond to an entity and each edge is a relationship between two entities. Applications can be modeled using several graph alternatives: (i) Simple graphs, (ii) Attributed graphs, and (iii) Multilayer networks.

A. Need for Multilayer Network Model

Multilayer networks provide an alternate model for representing complex datasets. It can capture each relationship as a layer which is a separate simple graph increasing understandability. It also affords itself for flexible analysis of layers. Nodes from different layers can also be connected by an edge if relationships exist between two entity types. For example, actor and director layers can be connected by the 'direct-actor' relationship where a director entity has an edge with each actor entity s/he has directed in a movie. The design of multilayer network model for a dataset is beyond the scope of this paper and can be found in [4]. Multilayer networks, by their structure, also offer flexibility to analyze each layer individually (and in parallel) and arbitrary subsets of layer combinations. Multilayer networks, based on entity types in each layer and connectivity within and across layers, can be classified into homogeneous MLNs (HoMLNs, Figure 1(a)), heterogeneous MLNs (HeMLNs, Figure 1(b)), and hybrid MLNs (HyMLNs, Figure 1(c)). Since the focus of this paper is HeMLNs, thus we start with its formal definition.

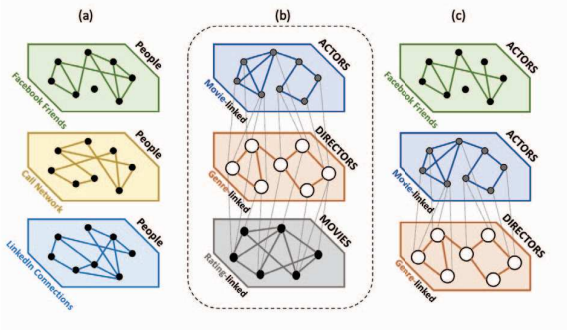


Figure 1: MLN Types

Definition 1. *A multilayer network $MLN(G, X)$, is defined by two sets of graphs. The set $G = \{G_1, G_2, \ldots, G_n\}$ contains **simple** graphs (one for each layer), where $G_i(V_i, E_i)$ is defined by a set of vertices V_i and a set of edges E_i. An edge $e(v, u) \in E_i$, connects vertices v and u, where $v, u \in V_i$. The set $X = \{X_{1,2}, X_{1,3}, \ldots, X_{m-1,m}\}$ consists of **bipartite** graphs[1]. Each graph $X_{i,j}(V_i, V_j, L_{i,j})$ is defined by two sets of vertices V_i and V_j and a set of edges (or links) $L_{i,j}$, such that for every link $l(a, b) \in L_{i,j}$, $a \in V_i$ and $b \in V_j$, where V_i (V_j) is the vertex set of graph G_i (G_j). Some of $X_{i,j}$ can be empty.*

Problem Addressed: Given an HeMLN with l layers – G_1

[1]The number of bipartite graphs can be less or even more than the number of graphs/layers. That is, m can be $<=$ or even $> n$.

979-8-3315-0882-1/24/$31.00 ©2024 IEEE
DOI 10.1109/ICKG63256.2024.00009

(V_1, E_1), G_2 (V_2, E_2), ..., G_l (V_l, E_l), where V_i and E_i are the vertex and edge set in the i^{th} layer and $X = \{X_{1,2}, X_{1,3}, \ldots, X_{n-1,n}\}$ are interlayer **bipartite** graphs – the goal is to discover substructures of the HeMLN for any r layers **correctly and efficiently** using the decoupling approach. For correctness, r HeMLN layers under consideration are aggregated into a simple graph using the Boolean OR-operator, and substructures discovered on that simple graph are used as Ground Truth (GT).

Use of Map/Reduce: We have used the Map/Reduce paradigm as an example of the distributed and parallel processing approach. Without loss of generality, any other paradigm (e.g., Spark) can be used in its stead without modifying the overall approach.

Contributions of this paper are:

- **Map/Reduce Algorithm** for substructure discovery in a layer.
- **Map/Reduce Composition algorithm** to correctly generate substructures spanning layers.
- Partitioned approach to **both** layer and interlayer graph processing.
- **Extensive experimental analysis** on a large number of synthetic graphs.
- **Accuracy** with ground truth, response time, and speedup comparisons.

Road map: MLN analysis alternatives are briefly discussed in Section II to highlight our choice. Section III has related work. Preliminaries and terminology are outlined in Section IV. Section V details the composition algorithm and Map/Reduce implementation of the proposed substructure discovery. Detailed experimental evaluation is given in Section VI. Section VII has conclusions.

II. MLN Analysis Alternatives and Challenges

Figure 2 shows three alternatives for performing analysis on MLNs. Figure 2(a) shows the traditional approach, where an MLN is conflated into a simple graph using type-independent [5] and projection-based [6] approaches. As both ignore type information for the transformation, they do not support structure and semantics preservation. As observed in the literature, *without additional mappings*, the above aggregation approaches are likely to result in some information loss and distortion of properties [7], or hide the effect of different entity types and/or different intra- or inter-layer relationships as elaborated in [8]. At the other end of the spectrum, Figure 2(c) shows computing the result by traversing the whole MLN as a single graph. Although this has been implemented for community detection (e.g., Infomap recently, [9]), this is likely to be computationally expensive and not flexible as the number of layers and data sizes become large and new MLN algorithms need to be developed for each analysis.

Figure 2(b), on the other hand, shows an approach (termed **networking decoupling**) where the analysis metric for each layer is computed independently (possibly in parallel) during the analysis (Ψ) phase **only once** and the results are composed using a binary operator Θ as shown [10], [11]. Other layer or interlayer information is **not used** while processing a layer! This approach has been shown to be effective, can be done for Boolean operations without aggregating and losing type information. Furthermore, it has been shown to be more efficient than the approaches shown in Figures 2(a) or (c).

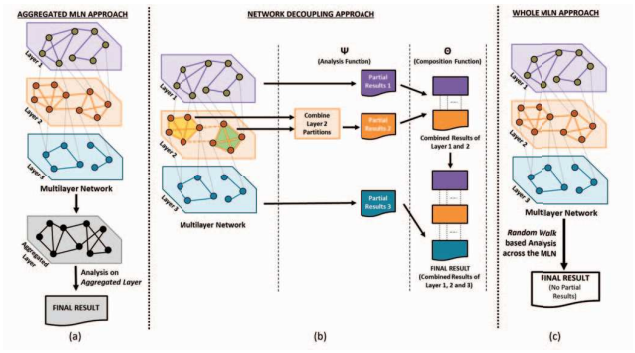


Figure 2: Lossy (a) Vs. Decoupling (b) Vs. MLN approaches

The challenges in developing a substructure discovery algorithm are: *i) enumerating all connected subgraphs of any size in a given graph or forest (completeness), ii) identifying duplicates, if any, and remove them (soundness), iii) count isomorphic substructures to apply the metric for each distinct substructure and rank them, and iv) retain top-k substructures for the next iteration. The enumeration is typically done iteratively increasing the connected subgraph size by 1 with each iteration. Evaluation of each substructure with the desired metric (frequency or minimum description length) is carried out after each iteration. To contain the search space, a heuristic is applied to carry top-k results from each iteration. Current algorithms do this on a simple graph or forest.* **How the decoupling approach has been adapted to the HeMLN substructure discovery is the focus of this paper and is elaborated in Section V.**

III. Related Work

SUBDUE was the earliest main memory algorithm [12] developed for substructure discovery. It can perform both supervised and unsupervised substructure discovery. It uses an iterative algorithm to systematically generate larger substructures which are evaluated using Minimum Description Length (MDL [13]). SUBDUE uses the notion of a *beam* to restrict the number of substructures carried to the next iteration.

AGM [14] and FSG [15] are two popular main memory graph mining algorithms that use the *apriori* concept. These approaches generate frequent (k+1) subgraphs from frequent k-subgraphs. The FSG identifies repeating substructures in graphs using an apriori approach [15]. This is different from Subdue since it entails finding interesting substructures in a graph or forest. Canonical labeling is added to the apriori algorithm. The property that identical graphs have identical canonical labeling has been strategically used to identify frequent substructures. FSG determines canonical labels using a flattened graph adjacency matrix.

Disk-based graph mining algorithms [16]–[18] were developed to deal with larger graph sizes. Portions of the graph are staged into memory for processing using buffer-management techniques. Indexing was used to improve retrieval for staging. gIndex [19] uses frequent substructures as units for indexing. However, these solutions need explicit data marshaling between disk and main memory entailing developing buffer management strategies. This leads to the use of relational databases for graph mining to leverage buffer management and query optimization. HDB-Subdue [20] used the Relational DBMS for graph representation and SQL to discover substructures. This approach was able to scale to handle graphs with millions of edges. However, self-joins on large relations and the number of joins needed seemed to have limited scaling further.

A number of graph mining methods have shown success in cloud-based deployments [21]. Splitting/partitioning large graphs into manageable chunks for distributed processing is explored in [22]. Scalable substructure discovery on large simple graphs using Map/Reduce has been developed in [23]. A graph can be split into partitions in different ways, processed by distributed and parallel architecture, and results from across partitions combined to obtain correct results. *This paper's focus is on MLNs instead of simple graphs. Further, the decoupling-approach is chosen which entails the development of a composition function for its efficient and flexible analysis. It also uses extant substructure discovery for each layer. The correctness of the composition algorithm is also validated by comparing results with the ground truth.*

IV. Preliminaries and Terminology

Edge List as Input: A graph (layer, interlayer, or a partition) is represented as a list of unordered edges. Each edge is completely represented by a 5 element tuple[2] $< E_l, V_{sid}, V_{sl}, V_{did}, V_{dl} >$ where, E_l is edge label, V_{sid} is source vertex ID, V_{sl} is source vertex label, V_{did} is destination vertex ID and V_{dl} is destination vertex label. In this context, it is important to note that the vertex IDs (V_{sid} and V_{did}) are guaranteed to be unique. However, it is not necessary for the vertex labels (V_{sl} and V_{dl}) and the edge label (E_l) to be unique. Table I shows the edge list representation for the input graph (or a layer) shown in Figure 3 (a).

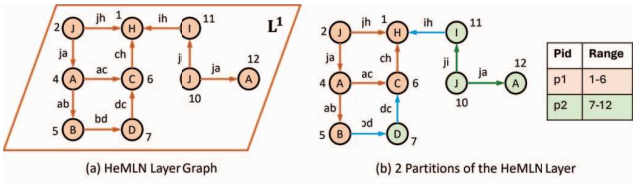


Figure 3: Partitions for a HeMLN layer graph

This 5 element tuple edge representation (that includes direction) is used to represent a k-edge substructure (a connected graph with k edges and $k+1$ nodes) as a collection of k 1-edge substructures. Our algorithm takes an input graph represented as a text file with a 1-edge substructure per line.

Adjacency List: Adjacency list is a representation where each vertex on which the edges are incident (both in and out) are associated with the node using a list (of 1-edge substructures.) This adjacency list is used for the expansion of a node in a subgraph to generate new substructures. Table II shows the adjacency list of the input layer graph shown in Figure 3 (a).

Table I: Edge List

Edge List
⟨ab,4,A,5,B⟩
⟨ac,4,A,6,C⟩
⟨bd,5,B,7,D⟩
⟨ch,6,C,1,H⟩
⟨dc,7,D,6,C⟩
⟨ja,2,J,4,A⟩
⟨jh,2,J,1,H⟩
⟨ih,11,I,1,H⟩
⟨ja,10,J,12,A⟩
⟨ji,10,J,11,I⟩

Table II: Adjacency List

Vertex ID	Adjacency List
1	⟨ch,6,C,1,H⟩;⟨jh,2,J,1,H⟩;⟨ih,11,I,1,H⟩;
2	⟨ja,2,J,4,A⟩;⟨jh,2,J,1,H⟩;
4	⟨ab,4,A,5,B⟩;⟨ac,4,A,6,C⟩;⟨ja,2,J,4,A⟩;
5	⟨ab,4,A,5,B⟩;⟨bd,5,B,7,D⟩;
6	⟨ac,4,A,6,C⟩;⟨dc,7,D,6,C⟩;⟨ch,6,C,1,H⟩;
7	⟨bd,5,B,7,D⟩;⟨dc,7,D,6,C⟩;
10	⟨ja,10,J,12,A⟩;⟨ji,10,J,11,I⟩;
11	⟨ih,11,I,1,H⟩;⟨ji,10,J,11,I⟩;
12	⟨ja,10,J,12,A⟩;

Table III: Adjacency List Partitions

Vertex ID	Adjacency List Partition for p1
1	⟨ch,6,C,1,H⟩;⟨jh,2,J,1,H⟩; ⟨ih,11,I,1,H⟩;
2	⟨ja,2,J,4,A⟩;⟨jh,2,J,1,H⟩;
4	⟨ab,4,A,5,B⟩;⟨ac,4,A,6,C⟩; ⟨ja,2,J,4,A⟩;
5	⟨ab,4,A,5,B⟩;⟨bd,5,B,7,D⟩;
6	⟨ac,4,A,6,C⟩;⟨dc,7,D,6,C⟩; ⟨ch,6,C,1,H⟩;

Vertex ID	Adjacency List Partition for p2
7	⟨bd,5,B,7,D⟩;⟨dc,7,D,6,C⟩;
10	⟨ja,10,J,12,A⟩;⟨ji,10,J,11,I⟩;
11	⟨ih,11,I,1,H⟩;⟨ji,10,J,11,I⟩;
12	⟨ja,10,J,12,A⟩;

Graph Partitioning: An MLN layer L_i can be partitioned into p partitions (L_1^i, L_2^i,..., L_p^i) for distributed processing. We use *range-based partitioning* [24] to create the partitions using vertex IDs. Each graph partition[3] is a range of node IDs and the size of each partition need not be same. There can be missing vertex IDs in a given range. As the ranges are disjoint, nodes in adjacency list partitions are also disjoint. Each vertex ID in the range and its adjacency list corresponds to a single adjacency list partition. If neighboring nodes are in two partitions, the edge connecting them will be in the adjacency list of both partitions. As a result, during expansion, the same substructure can belong to many graph partitions. As adjacency list partitions are indexed on vertex IDs, each substructure is expanded only once.

Two partitions of the graph in Figure 3 (a) are shown in Figure 3 (b). Partition $p1$ is assigned vertex IDs from 1 to 6, whereas partition $p2$ is allotted vertex IDs 7 to 12 (see table in Figure 3.) Both partitions are connected by blue edges, which appear in different adjacency list partitions. The adjacency list partitions are displayed in table III. The edges that occur in both adjacency list partitions are $< bd, 5, B, 7, D >$, $<dc, 7, D, 6, C>$, and $<ih, 11, I, 1, H>$. If these two partitions belong to two layers, these edges will become interlayer edges.

[2]This representation is generic and can be extended to attributed graphs by using a distinct edge identifier in the edge representation. Our discussion excludes multiple edges.

[3]Graph partition and partition are used interchangeably. Adjacency List partition is an adjacency list for a specific graph partition.

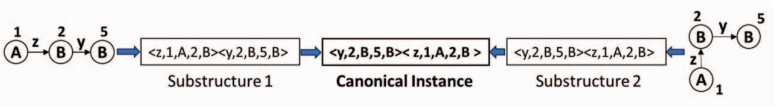


Figure 4: Duplicate substructure identification using canonical instances

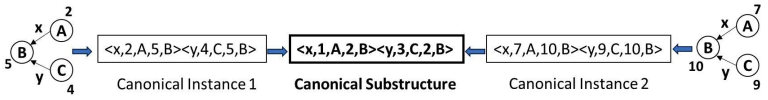


Figure 5: Graph isomorphism and canonical substructures

Independent Expansion of Substructures: For substructure discovery, substructures[4] of progressively increasing size are generated systematically in each iteration. As the best substructures (for a given metric) can be of any size, all sizes (or as specified by parameters) need to be generated. Hence, graph expansion plays a salient part in the process of discovering the best substructures. A k-edge substructure is expanded in the k^{th} iteration to general all (k+1)-edge substructures. **Independent** expansion is done on each substructure and on each node of every substructure using the appropriate adjacency list partitions. As this unconstrained expansion generates an unwieldy search space, a *beam* is used to select top-*beam* substructures in each iteration to use in the next iteration. Also, as this **independent** expansion results in the generation of duplicate substructures *within an iteration*, care has to be taken to remove duplicates in each iteration before applying the metric.

Canonical Instance for Duplicate Elimination: Independent expansion of substructures, as noted earlier, can lead to duplicate substructure instances. By a duplicate, we mean that the same substructure instance may be generated more than once during the expansion of *different* substructures in the same iteration. Lexicographic ordering (or canonical form) of substructure representation is used to identify duplicates. Edges in a substructure are ordered based on edge label, then source vertex label, then destination vertex label, and finally source and destination vertex IDs. If any of the values match, the comparison moves forward to the next component, else the ordering is performed. A substructure can be uniquely represented using the lexicographic order of 1-edge components. This is called a canonical k-edge instance. Intuitively, two duplicate k-edge substructures must have the same ordering of labels and vertex IDs when converted to canonical k-edge instance. Figure 4 shows an example of duplicate substructures and how duplicates have the same canonical instance. Note that substructure $< z, 1, A, 2, B >$ on the left has been expanded with edge $< y, 2, B, 5, B$ and substructure on the right $< y, 2, B, 5, B >$ has been expanded with the edge $< z, 1, A, 2, B >$. All instances generated in each iteration are converted to canonical form to eliminate duplicates.

[4]Substructures and substructure instances are used interchangeably. Substructure instances contain vertex IDs from the graph. These are termed canonical instances when the edges are ordered lexicographically. We also refer to substructures which are exact isomorphs that are different from substructure instances. These are referred to as canonical substructures where relative ordering instead of lexicographic ordering of node IDs is used.

Canonical substructures for identifying Substructure Isomorphs: Exact isomorphs in a graph have the same structure in terms vertex and edge labels as well as connectivity, but differ in vertex IDs, in contrast to duplicates. After duplicate elimination, we identify isomorphs to count their occurrences. For this we need to convert canonical instances of substructures to canonical substructures using relative ordering of vertex IDs. Intuitively, in the canonical form, two isomorphic substructures have the **same relative ordering** of vertex numbers. Conversion to canonical substructure is done by replacing each vertex ID with their relative positions in the instance starting from 1. The inclusion of these relative positions is critical for differentiating the connectivity of the instances. Figure 5 shows an example of how canonical substructure is created from the canonical instance. It can be seen that the isomorphs have different canonical instances. Using the above technique, the relative positioning of vertex IDs (2, 5, 4) for the canonical instance 1 and (7, 10, 9) for the canonical instance 2 are converted to (1, 2, 3). Hence we can identify isomorphs using canonical substructures.

A. *Metrics Used for Ranking Substructures*

The Minimum Description Length is an information-theoretic, domain-independent metric that has been demonstrated to emphasize the significance of a substructure in terms of its ability to compress a complete graph or forest. Although it is defined, originally, in terms of bits used for graph representation, we use the number of nodes and edges for that purpose. The general formula for MDL (DL(G)/(DL(S) + DL(G|S))) represents the description length of the substructure S being evaluated, DL(G|S) represents the description length of the graph G when compressed by replacing each instance of the substructure S as a node, and DL(G) (or DL(S)) represents the description length of the original graph (or the substructure S.) The substructure of the graph achieves the highest compression when the MDL value is the highest. Both the frequency of the subgraph and its connectivity have an impact on compression. The frequency of substructures can also be used as a metric as used in FSG and others.

V. Composition Algorithm and Architecture

Details of using the decoupling approach for iterative substructure discovery of HeMLNs are shown in Figure 6. During the k^{th} iteration, substructures of size k from each layer[5] are

[5]For simplicity of discussion, two layers are shown. The algorithm and the architecture work for more than two layers.

expanded to generate all substructures of size $k+1$. A layer may be partitioned for distributed and parallel processing for which range partitioning is used and partitioned substructure discovery algorithm proposed in [24] is used. The composition function takes generated substructure instances (after duplicate elimination for each layer) along with interlayer edges to generate substructure instances that span 2 layers that could not be generated during the analysis phase of a layer as no information is used other than the layer itself. Duplicates are eliminated for the composed substructures. After duplicate elimination, **all** substructures (from layer 1, layer 2, and composed substructures) are grouped, and canonical instances are relatively ordered for isomorphic substructure counting. Then the metric is applied and each substructure is ranked. Top-beam substructures (and their instances) are used for the next iteration. The termination condition is checked after each iteration, which is based on the maximum substructure size.

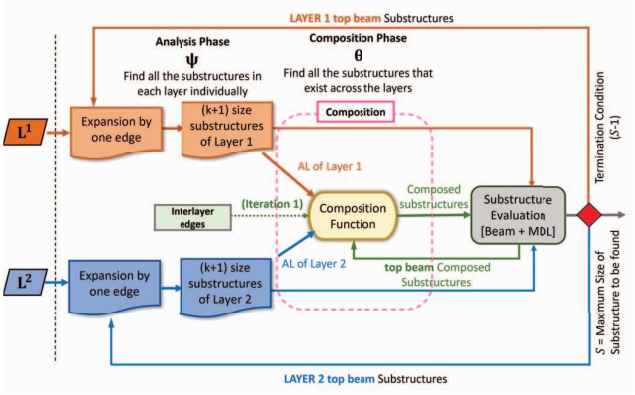


Figure 6: Iterative Decoupling-Based Substructure Discovery in HeMLN (k^{th} iteration)

Composition algorithm He-ICA is shown as Algorithm 1 after introducing general notations used in Table V and the adjacency lists used during composition in Table IV.

Table IV: Adjacency Lists used for Composition

Adj List	Description
L^1_{AL}	Initially generated L^1 adj list for iteration $k=1$
L^2_{AL}	Initially generated L^2 adj list for iteration $k=1$
$IL^{1,2}_{AL}$	Adj list of interlayer edges for iteration $k=1$
$L^1_{AL^k}$	Adj list generated from k–edge instances of L^1 for iteration k where $k=2,3,\ldots,S-1$ and S is given max
$L^2_{AL^k}$	Adj list generated from k–edge instances of L^2 for iteration k where $k=2,3,\ldots,S-1$ and S is given max
$IL^{1,2}_{AL^k}$	Adj list generated from top BEAM k–edge interlayer instances in previous $(k-1)$th iteration for iteration k where $k=2,3,\ldots,S-1$ and S is given max

As shown in Algorithm 1, the input to our composition algorithm is the list of k-edge composed instances $IL^{i,j}_k$, adjacency list of composed instances generated in previous iteration $IL^{i,j}_{AL^k}$, and adjacency list of i^{th} and j^{th} layer generated from k-edge expanded and not-expanded instances of i^{th} and j^{th} layer respectively ($L^i_{AL^k}$ and $L^j_{AL^k}$ respectively). The output generated is the $(k+1)$-edge composed substructures of k^{th} iteration.

Table V: Table of Notations

Notation	Description
k	Used as subscript for iteration and takes values $1,2,\ldots,s-1$ where s is size of substructures to be obtained.
i,j	Used for indicating interlayer -ids, where $i=1,2,\ldots,n$ and $j=i+1$ where n is total number of layers in HeMLN.
$IL^{i,j}_k$	Set of composed interlayer substructures of i,j and this changes with iteration. For $k=1$, it is list of interlayer edges.
$IL^{i,j}_{AL^k}$	Adjacency list of composed substructures of previous iteration and this changes with iteration. For $k=1$, its adjacency list of interlayer edges.
$L^i_{AL^k}$	Adjacency list of expanded substructures of i-th layer and this changes with iteration.
$L^j_{AL^k}$	Adjacency list of expanded substructures of j-th layer and this changes with iteration.
ks	Each $ks \in IL^{i,j}_k, ks =< E^1_l, V^1_{sid}, V^1_{sl}, V^1_{did}, V^1_{dl} > ;< E^2_l, V^2_{sid}, V^2_{sl}, V^2_{did}, V^2_{dl} >;\ldots;< E^k_l, V^k_{sid}, V^k_{sl}, V^k_{did}, V^k_{dl} >$ as list of 5 tuples where, E_l- edge label, $V_{sid}-$ source vertex-id, $V_{sl}-$ source vertex label, $V_{did}-$ destination vertex-id, $V_{dl}-$ destination vertex label.
ci	Canonical instance, generated after arranging the expanded instance in *lexicographical order*.

Algorithm 1 Composition Algorithm He-ICA (k^{th} iteration)

Require: $IL^{i,j}_k$, $IL^{i,j}_{AL^k}$, $L^i_{AL^k}$, $L^j_{AL^k}$
Ensure: Return Top *beam* substructures of size *k+1*
1: $IL^{i,j}_{k+1} \leftarrow \emptyset$
2: **for** each k-edge instance $ks \in IL^{i,j}_k$ **do**
3: **for** each vertex-id $v \in ks$ **do**
4: $EL_v \leftarrow \{v \in IL^{i,j}_{AL^k} \cup L^i_{AL^k} \cup L^j_{AL^k} | v.edgelist\}$
 {edge list of v from union of adjacency lists}
5: **for** each edge $e \in EL_v$ **do**
6: **if** $e \notin ks$ **then**
7: $ci \leftarrow merge\ ks$ to e in lexicographical order
8: **if** $ci \notin IL^{i,j}_{k+1}$ **then**
9: $IL^{i,j}_{k+1} \leftarrow IL^{i,j}_{k+1} \cup \{ci\}$
 {check for duplicates in the result set}
10: **end if**
11: **end if**
12: **end for**
13: **end for**
14: **end for**

For each of the k-edge instances, we expand on all the vertex IDs present in the instance using all the three adjacency lists as depicted (**lines 2 to 14.**) We get all the edge lists corresponding to the vertex ID from the union of all the three adjacency lists as depicted in (**lines 2 to 4.**) We expand each edge on vertex ID v and convert to *canonical instance* using *lexicographical order technique*. This helps us to eliminate duplicates as shown in (**lines 5 to 9**.) Hence, all the missing substructures that span layers under consideration are generated using interlayer substructures and layer substructures.

A. Distributed Architecture

Map/Reduce architecture is used for the implementation, but this can be any other distributed architecture without the need

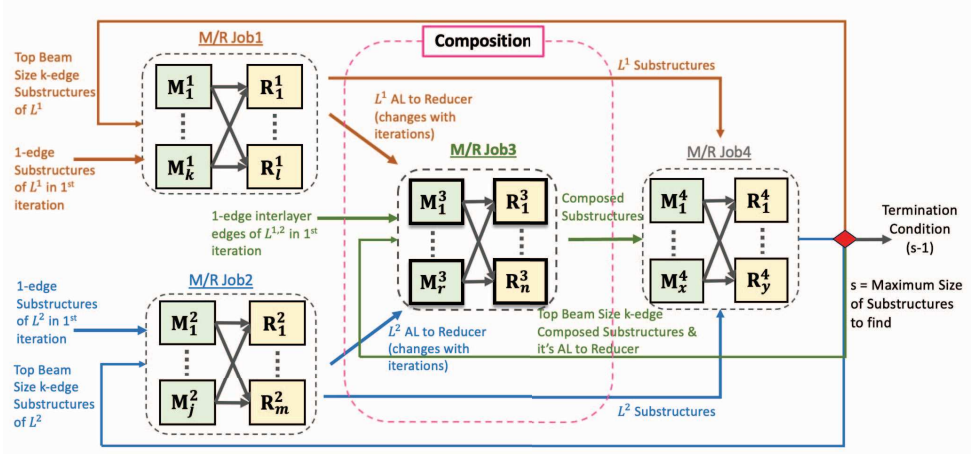


Figure 7: Complete Map/Reduce Architecture and Data Flow (k^{th}iteration)

to change the overall approach sans minor details (e.g., no need to load adjacency lists as they can be maintained in memory across iterations if Spark is used.) The complete Map/Reduce architecture and data flow is shown in Figure 7 including the flow of results from one job to the next (for k^{th} iteration.) The first and second map/reduce jobs are utilized for performing layer-wise expansion, which is the independent expansion of subgraphs using the adjacency list for that partition.

In our composition, we employ the adjacency list of interlayer edges during the first iteration. Beyond that we use the adjacency list generated in the previous iteration using the composed substructures. In addition, we generate the substructure within the interlayer by exclusively utilizing interlayer edges in the first iteration, and composed instances in subsequent iterations for expansion purposes.

All the substructures from layer 1, layer 2, and the composed substructures in job 3 are used in job 4 for substructure evaluation. We rank the substructures based on MDL metric and top-beam substructures are carried to the next iteration for further expansion. This process continues until the termination condition is satisfied.

VI. Experimental Evaluation And Validation

In this section, we describe the Map/Reduce environment, generation of synthetic HeMLN layers, and validation of correctness using ground truth (GT). In addition to that, we analyze a large number of synthetic datasets with varying graph characteristics and map/reduce configurations. Table VI shows the Expanse cluster (located at San Diego super-computing center (SDSC)) details used for experiments.

Table VI: Expanse Cluster Details

What	SDSC Expanse Configuration
SSCUs	13 SDSC Scalable Compute Units
Node Count	728
Cores/Node	128 built on 2 processors (64 cores each)
Processor	AMD EPYC 7742
Memory/Node	256 GB DDR4 DRAM
Total Storage	1TB Intel P4510 NVMe PCIe SSD

Dataset Generation: For our experiments, we use a single graph to generate as many layers of HeMLN as needed. This allows us to generate layers with diverse characteristics in terms of nodes, edges, and interlayer edges. This also allows us to test the correctness using the GT. We embed substructures (of different sizes with differing frequencies) for testing the correctness of our algorithms.

Dataset Description: Synthetically generated datasets are used to cover diverse input MLN characteristics. For each graph, we further generated three random node distributions that have an effect on intra and interlayer edges as well as embedded substructure distribution. Table VII shows the datasets used, their purpose, and M/R configurations used for analysis. For these datasets, diverse two layer HeMLNs

Table VII: Dataset Description

Dataset	Used For	Layer M/R configs
Synthetic: 50KV_100KE	Accuracy, Response Time	2M/2R, 4M/4R 8M/8R, 16M/16R
Synthetic Large: 1MV_4ME,2.5MV_10ME	Response Time, Speedup	16M/16R, 32M/32R 64M/64R, 128M/128R

were generated with three node distributions (50/50, 70/30, and 90/10, nodes chosen randomly) for testing. An edge connecting two layers is an interlayer edge. Nodes, edges, and interlayer edges for each layer are also shown in Table VIII. **Dataset size nnnKV_eeeME denote nnn Kilo vertices and eee Million edges**.

A. Correctness Validation using Ground Truth.

We included multiple embedded graphs of different sizes and frequencies to verify our algorithm and framework with ground truth generated by Subdue. We could only do it for graphs up to 100K edges. For larger datasets, we also embedded graphs of known size and frequency to validate our results directly without using ground truth. This comprehensive testing ensured a principled evaluation.

We conducted experiments using synthetic graphs generated by Subgen [25]. Subgen is used to generate synthetic graphs of

Table VIII: Datasets With Node Distributions

Dataset	Node Dist.	L^1 #Nodes	L^2 #Nodes	L^1 #Edges	L^2 #Edges	$L^{1,2}$ #Edges
50KV 100KE	50/50	25000	25000	29236	29048	49858
	70/30	35000	15000	49250	8880	41870
	90/10	45000	5000	81203	1040	17757
100KV 500KE	50/50	50000	50000	124719	124837	250444
	70/30	70000	30000	245122	45196	209682
	90/10	90000	10000	404942	4998	90060
400KV 1ME	50/50	200000	200000	250297	249961	499742
	70/30	280000	120000	490686	89987	419327
	90/10	360000	40000	809920	9967	180113
800KV 3ME	50/50	400000	400000	749725	750265	1500010
	70/30	560000	240000	1468707	269877	1261416
	90/10	720000	80000	2429768	29972	540260
1MV 4ME	50/50	500000	500000	998911	1000756	2000333
	70/30	700000	300000	1959976	359709	1680315
	90/10	900000	100000	3239820	40233	719947
2.5MV 10ME	50/50	1250000	1250000	2497921	2498771	5003308
	70/30	1750000	750000	4902860	899571	4197569
	90/10	2250000	250000	8100658	99761	1799581

given node and edge distributions, and is embedded with substructures with some frequency. We have generated synthetic graphs ranging from 50KV_100KE to 2.5MV_10ME sizes.

Table IX: Ground Truth Comparison Using SUBDUE for 50KV_100KE dataset with 5- and 10-edge embedded substructures

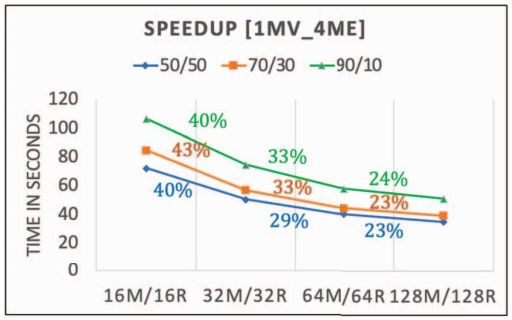
		5-edge	10-edge
Embedded Frequency (Ground Truth)		**3000**	**1000**
Node Distribution (L1/L2)	Substructure	Instance	Frequency
50/50	L1	41	17
	L2	40	16
	Composed	2919	967
	Total	**3000**	**1000**
70/30	L1	342	178
	L2	1	2
	Composed	2657	820
	Total	**3000**	**1000**
90/10	L1	1416	457
	L2	0	0
	Composed	1584	543
	Total	**3000**	**1000**

Layer Node Distribution And GT Accuracy. Table IX shows GT substructures identified by our algorithm for different node distributions. Total substructures found in layer 1, and layer 2, along with the total composed substructure by the He-ICA algorithm are shown. For 50/50 node distribution, there are significantly more substructures that span layers (and are detected as part of the composition) and less in each layer, but all add up to the embedded frequency. The number of embedded substructures increases as layer distribution changes from 50% to 70% and increases even further when that percentage increases to 90%. The same observation is repeated for the larger embedded 10-edge substructure. Correctness with GT is established in both cases when node distribution is changed. **Substructure and frequency correctness was established for larger synthetic datasets, indicating that He-ICA correctly generated all substructures that span layers for different node distributions**.

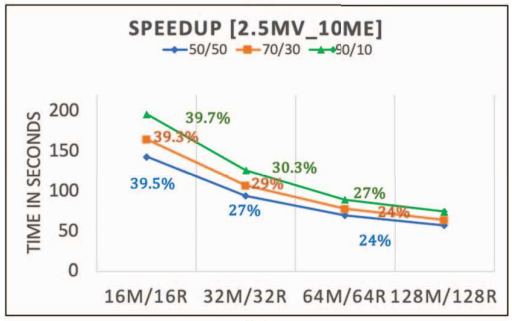
B. Response Time and Speedup on Large graphs.

To evaluate our response time, we use different number of partitions alongside an equivalent number of mappers and reducers to maximize parallelization. The goal is to observe a proportionate decrease in the total response time as we scale up resources. We seek to comprehend the speedup characteristic to understand whether it adheres to a linear trend or demonstrates diminishing returns beyond a certain configuration. Our verification of the speedup and response time of our approach involves the use of large synthetic datasets.

Synthetic Graphs: For the analysis of speedup achieved on synthetic datasets, we explored various dataset sizes, ranging from 50,000 vertices and 100,000 edges (50KV_100KE) to 2.5 million vertices and 10 million edges (2.5MV_10ME). Despite variations in layer size, each layer was partitioned into the same number of partitions. We conducted experiments with 2, 4, 8, 16, 32, 64, and 128 partitions, maintaining an equal number of mappers and reducers for each configuration. Although consistent trend was observed across all datasets, we limit, due to space constraints, our focus to the analysis of the largest datasets, namely, 1M_4ME and 2.5MV_10ME.



(a)



(b)

Figure 8: Speedup: (a) 1MV_4ME, (b) 2.5MV_10ME

As we can observe in Figure 8(a), we get an average speedup of 30% by increasing both the number of partitions and the number of mappers/reducers from 16 to 32. Subsequently, a speedup of 22% was noted upon further increasing them from 32 to 64. Finally, a speedup of 13% was observed by further increasing them from 64 to 128. It is important to highlight that the achieved speedup was not

linear; doubling the mappers and reducers did not result in halving the time taken. As the number of partitions increased, the speedup exhibited diminishing returns. Similar trend was observed in Figure 8(b). We have also studied the partitions based on node distributions and observed better speedup as more partitions and more resources on smaller layers incurred additional overhead.

C. *Response Time of All Datasets*

To provide a comprehensive overview, we have synthesized the results from various experiments into a single graph in Figure 9.

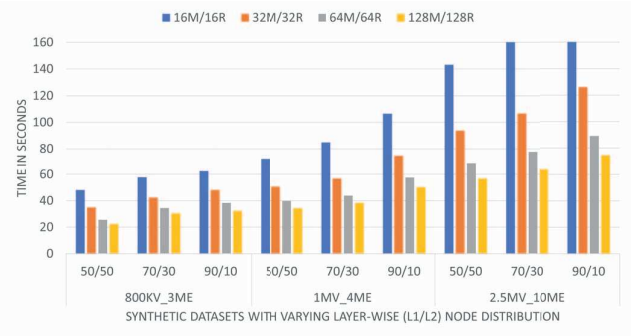


Figure 9: Response Times of Synthetic Datasets with Layer Node Distributions and M/R configurations

As can be consistently seen from Figure 9, the more skewed the layer node distribution is, greater is the processing time, as the processing time is dominated by the denser layer. However, there is significant speedup as resources are doubled. We believe that this can be further improved by choosing resources to match layer edge and interlayer edge sizes.

D. *Desiderata*

We have analyzed more real-world and synthetic datasets than able to include in the paper. We have verified the correctness of composition on forests in layers, GT substructures spanning layers, and more GT validations of He-ICA with SUBDUE using pathological cases have been done successfully.

VII. Conclusions and Future work

This paper introduced a new algorithm for substructure discovery in Heterogeneous Multilayer Networks (HeMLNs) using the decoupling-based approach. We designed and implemented a composition algorithm to identify missing substructures that span HeMLN layers. We have also cast the algorithm into distributed and parallel processing paradigm for scalability. We validated the correctness of our algorithm using GT and conducted extensive experimental analyses on synthetic datasets.

Substructure discovery on MLNs offers plenty of challenges and opportunities for future research.

Acknowledgments

This work was partially supported by NSF awards #1955798 and #1916084.

References

[1] A. P. F. Miguel E. Coimbra and L. Veiga, "An analysis of the graph processing landscape," *Journal of Big Data*, 2021.
[2] N. G. de Bruijn, "A combinatorial problem," *Proc. Akad. Wet. Amsterdam*, vol. 49, pp. 758–764, 1946.
[3] A. T. Balaban, "Applications of graph theory in chemistry," *Journal of Chemical Information and Computer Sciences*, vol. 25, no. 3, pp. 334–343, 1985. [Online]. Available: https://doi.org/10.1021/ci00047a033
[4] A. Santra, K. S. Komar, S. Bhowmick, and S. Chakravarthy, "From base data to knowledge discovery - *A life cycle approach* - using multilayer networks," *Data Knowl. Eng.*, vol. 141, p. 102058, 2022.
[5] M. D. Domenico, V. Nicosia, A. Arenas, and V. Latora, "Layer aggregation and reducibility of multilayer interconnected networks," *CoRR*, vol. abs/1405.0425, 2014.
[6] A. Berenstein, M. P. Magarinos, A. Chernomoretz, and F. Aguero, "A multilayer network approach for guiding drug repositioning in neglected diseases," *PLOS*, 2016.
[7] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter, "Multilayer networks," *CoRR*, vol. abs/1309.7233, 2013.
[8] M. De Domenico, A. Solé-Ribalta, S. Gómez, and A. Arenas, "Navigability of interconnected networks under random failures," *Proc. of Ntl. Acad. of Sciences*, 2014.
[9] J. D. Wilson, J. Palowitch, S. Bhamidi, and A. B. Nobel, "Community extraction in multilayer networks with heterogeneous community structure," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 5458–5506, 2017.
[10] A. Santra, S. Bhowmick, and S. Chakravarthy, "Hubify: Efficient estimation of central entities across multiplex layer compositions," in *IEEE ICDM Workshops*, 2017.
[11] ——, "Efficient community re-creation in multilayer networks using boolean operations," in *Int. Conf. on Computational Science*, 2017.
[12] D. J. Cook and L. B. Holder, "Substructure discovery using minimum description length and background knowledge," *J. Artif. Intell. Res.*, vol. 1, pp. 231–255, 1994.
[13] J. Rissanen, "A universal prior for integers and estimation by minimum description length," *Annals of Statistics*, vol. 11, pp. 416–431, 1983.
[14] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in *Very Large Data Bases*, 1994, pp. 487–499.
[15] M. Deshpande, M. Kuramochi, and G. Karypis, "Frequent Sub-Structure-Based Approaches for Classifying Chemical Compounds," in *IEEE International Conference on Data Mining*, 2003, pp. 35–42.
[16] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan,: mining sequential patterns efficiently by prefix-projected pattern growth," in *ICDE*, 2001, pp. 215–224.
[17] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognition Letters*, vol. 19, pp. 255–259, 1998.
[18] S. Alexaki, V. Christophides, G. Karvounarakis, and D. Plexousakis, "On Storing Voluminous RDF Descriptions: The Case of Web Portal Catalogs," in *Intl. Workshop on the Web & Databases*, 2001, pp. 43–48.
[19] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, pp. 335–346.
[20] S. Padmanabhan and S. Chakravarthy, "Hdb-subdue: A scalable approach to graph mining," in *DaWaK*, 2009, pp. 325–338.
[21] S. Suri and S. Vassilvitskii, "Counting triangles and the curse of the last reducer," in *World Wide Web Conference Series*, 2011, pp. 607–614.
[22] S. Yang, X. Yan, B. Zong, and A. Khan, "Towards effective partition management for large graphs," in *SIGMOD*, 2012, pp. 517–528.
[23] S. Das and S. Chakravarthy, "Duplicate reduction in graph mining: Approaches, analysis, and evaluation," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 8, pp. 1454–1466, 2018.
[24] ——, "Partition and conquer: Map/reduce way of substructure discovery," in *International Conference on Big Data Analytics and Knowledge Discovery*. Springer, pp. 365–378.
[25] "http://ailab.wsu.edu/subdue."