RASMA: A Reverse Search Algorithm for Mining Frequent Subgraphs

Saeed Salem¹*, Mohammed Alokshiya and Mohammad Al Hasan²

Abstract

Background: Mining frequent co-expression networks enables the discovery of interesting network motifs that elucidate important interactions among genes. Such interaction subnetworks have been shown to enhance the discovery of biological modules and subnetwork signatures for gene expression and disease classification.

Results: We propose a reverse search algorithm for mining frequent and maximal subgraphs over a collection of graphs. We develop an approach for enumerating connected edge-induced subgraphs of an undirected graph by using a reverse-search algorithm, and then use this enumeration strategy for mining all maximal frequent subgraphs. To overcome the computationally prohibitive task of enumerating all frequent subgraphs while mining for maximal subgraphs, the proposed algorithm employs several pruning strategies, which substantially improve its overall runtime performance. Experimental results show that on large gene coexpression networks, the proposed algorithm efficiently mines biologically relevant maximal frequent subgraphs.

Conclusion: Extracting recurrent gene coexpression subnetworks from multiple gene expression experiments enables the discovery of functional modules and subnetwork biomarkers. We have proposed a reverse search algorithm for mining maximal frequent subnetworks. Enrichment analysis of the extracted maximal frequent subnetworks reveals that subnetworks that are more frequent are more likely to be enriched with biological ontologies.

Keywords: Biological Networks; Subgraph Enumeration; Frequent Subgraphs; Maximal Subgraphs; Reverse Search

Background

Advances in genome technologies allow for the probing of thousands of genes at the same time through the use of mRNA sequencing and gene expression microarray. Gene expression analysis aims at discovering gene clusters that have similar expression profiles and dysregulated genes that can be used as markers for solving various disease classification tasks. However, research has revealed that genes do not work in isolation and often a single gene does not have an independent effect on a phenotype, rather multiple genes interact together to control that phenotype. To capture such correlation among genes, gene coexpression networks have been proposed [1]. In coexpression networks analysis, a gene expression dataset is represented as a coexpression network where nodes represent genes and a link exist between a pair of genes if they exhibit significant correlation in the microarray analysis [2, 3]. Traditionally a single gene expression dataset was analyzed independently. However,
functional annotation and biological inference based on a single gene coexpression dataset has limitations due to experimental noise [2]. Learning from multiple gene expression datasets alleviates the noise problem. So, recent research has focused on mining biologically interesting gene coexpression subnetworks from multiple heterogeneous gene expression datasets. A set of genes that have similar expression profiles in multiple experiments is more likely to represent a biological module [1, 2]. The integrative analysis of multiple gene expression datasets enables the discovery of significant interactions involved in complex biological processes, and has been employed for functional annotation [1], active module discovery [4], and biomarker discovery [5]. An approach to identify these co-expression subnetworks is to mine frequent subgraphs over multiple gene expression networks. Careful study of these frequent subgraphs can lead to the identification of functional modules and the discovery of significant genes interactions playing key roles in complex diseases [6].

Existing algorithms for mining significant patterns from coexpression networks mainly follow network clustering, pattern enumeration approaches, or a combination of both. Lee et. al [1] proposed an approach that constructs an aggregate summary network by combining all the gene coexpression networks and removing coexpression links that appear in a small number of networks. Dense subnetworks are then extracted from the aggregate summary networks using a traditional module finding algorithm. Mining dense modules from the summary network results in false positive modules that are highly dense in the summary network but not dense in the individual networks as the links are scattered across the networks.

Other approaches followed a two-step approach for mining modules from multiple networks. Authors of [2] proposed CODENSE algorithm that mines dense modules from the aggregate network and then uses the similarity between edges’ occurrences to extract dense coherent modules. The output of the CODENSE algorithm is dense modules whose edges appear in similar graphs. [7] proposed an approach that first clusters edges that appear in similar graphs. The subnetwork induced by this edge cluster can be disconnected and sparse and thus the subnetwork is then partitioned into dense subnetworks. Since the edges in the cluster appear in similar sets of graphs, the dense subnetworks extracted from the edge cluster are coherent. In an earlier work, we proposed a similar approach that first mines maximal frequent edgesets and then extracts $k$-cliques and percolated $k$-cliques from these edgesets in the aggregate network [8]. Other approaches have been proposed to extract modules from a summary graph constructed by combining the edge topological similarity in the aggregate graph and edge occurrence similarity [9, 10].

A key limitation of all two step methods is the appearance of false-positive or false-negative edges in the network motifs; so frequent subgraph based algorithms are increasingly being used for mining co-expression networks. However, co-expression networks are generally large, sometimes having tens of thousands vertices, and traditional frequent subgraph mining algorithms [11, 12, 13, 14], which solve the subgraph isomorphism exactly are computationally prohibitive for mining from a collection of co-expression networks. However, a key observation that the vertex labels of a co-expression network are distinct, as each vertex represents a distinct gene, leads to efficient subgraph mining algorithms for co-expression networks—this has been first exploited by Koyuturk et al [15]. A major challenge for subgraph enumeration algorithms is the development of efficient traversal of the frequent subgraphs
enumeration tree, and avoiding the generation of duplicate frequent subgraphs. A reverse search algorithm defines a set of rules for generating frequent subgraphs from a parent frequent subgraph. These rules are employed for constructing the subgraph enumeration tree starting from single edges and following a depth-first pattern growth approach.

In this paper, we propose a novel reverse search algorithm for enumerating all edge-induced connected subgraphs of a graph. The reverse search utilizes the shortest distance between edges to check for valid subgraph extensions. Building on this enumeration approach, we propose an algorithm for mining all frequent and maximal frequent subgraphs from a graph database. To efficiently mine all maximal frequent subgraphs, we propose two pruning techniques that eliminate futile search subtrees in the frequent subgraph enumeration tree. These pruning strategies result in significant improvement in the running time of the algorithm. We demonstrate the effectiveness of the proposed algorithms with the pruning strategies on gene coexpression graphs, and show that the proposed algorithm is orders-of-magnitude faster than existing algorithms.

Related Work

Several algorithms have been proposed for mining frequent subgraphs from non-uniquely labeled graphs where multiple nodes in the same graph can be assigned the same label, e.g., atom labels in chemical compounds. These mining algorithms include mining all frequent subgraphs from multiple graphs [11, 12, 13, 14] and algorithms for mining a summary of all frequent subgraphs such as mining closed frequent subgraphs [16] and maximal frequent subgraphs [17]. Mining the complete set of frequent subgraphs is computationally intractable for large graph sets and small support thresholds and sampling methods for mining representative subgraphs have been proposed [18]. Mining subgraphs with certain topological properties has received attention and several algorithms have been proposed for mining subgraphs that are highly-connected in multiple graphs [19, 20, 21].

A special class of graphs is the graphs with unique-label nodes, e.g., gene coexpression networks, where no two nodes in the same graph have the same label. The problem of mining frequent subgraphs from graphs with unique vertex labels has received less attention. Though possible, it is not computationally efficient to employ existing subgraph mining algorithms designed for the general case to mine uniquely-labeled graphs. The computationally-intensive procedures of subgraph and graph isomorphism are not required for mining uniquely-labeled graphs. Moreover, the tasks of candidate generation and pruning is much simpler for uniquely-labeled graphs.

The backbone of mining subgraph algorithms is the enumeration strategy employed for enumerating all connected subgraphs as potentially all connected subgraphs could be frequent. Frequency and topological constraints are then enforced while enumerating the subgraphs. In sparse graphs, the number of connected subgraphs is much smaller than the number of all subgraphs. Moreover, the number of subgraphs that satisfy the frequency or topological constraints is much smaller than the number of connected subgraphs.
Koyuturk et al. [15] proposed the MULE (Mining Uniquely Labeled Edgesets) algorithm for mining frequent subgraphs. Moreover, an extension to the MULE algorithm was proposed to mine the closed and maximal subgraphs. At the core of the MULE algorithm is an enumeration approach for visiting all connected edge-induced subgraphs of a graph. The enumeration approach in the MULE algorithm visits each subgraph in the enumeration tree only once. It does so by defining the set of candidates to be explored at each step based on the set of edges visited and the current pattern. In the MULE algorithm, at each search node in the search space, the set of subgraphs generated from a given subgraph is not always the set of all supergraphs of a given subgraph because the missing supergraphs would be visited from other subgraphs. Because the frequency constraint satisfies the downward closure (Apriori) property, the minimum support constraint is enforced while traversing the subgraph lattice and a futile search branch is pruned once an infrequent subgraph is encountered. The number of frequent subgraphs in a graph dataset is very large, especially for small support thresholds. This is mainly because all the subgraphs of a frequent subgraph are also frequent (downward closure property). For downstream analysis of these frequent subgraphs, it is often desired to mine a representative set of these frequent subgraphs. Moreover, it can be computationally feasible to mine these representative patterns when mining all frequent subgraphs is not possible.

Several approaches have been proposed to mine succinct set of frequent subgraphs, including maximal frequent and close frequent subgraphs [22]. A frequent subgraph that does not have any frequent supergraph is a maximal frequent subgraph. For maximal frequent subgraphs, if a frequent subgraph does not have any frequent supergraph in the enumeration tree, then it is locally maximal frequent. The MULE algorithm checks if the the locally maximal subgraph is a subgraph of an already mined maximal frequent subgraph to ensure that it is a maximal subgraph. The set of discovered maximal subgraphs that has to be kept in memory can be very large and thus checking if a subgraph is a subgraph of an already discovered maximal subgraph can be computationally expensive. Another limitation of the MULE algorithm is that it does not have pruning strategies that eliminate the traversal of search branches that would result in locally maximal subgraphs that are not globally maximal frequent subgraphs. For the special case when the graph dataset has a single graph, and minimum support of 1, the MULE algorithm enumerates all connected edge-induced subgraphs of the single graph while in fact there is only one maximal subgraph.

Another approach for enumerating all connected subgraphs was proposed in [23]. The main idea of the approach is that for a given edge, the set of all connected subgraphs can be partitioned into two groups: the subgraphs that have the edge, and the subgraphs that do not have the edge. The recursive algorithm has an amortized computation time of $O(1)$ for each subgraph.

Reverse Search is a recent search approach for enumeration problems [24]. The basic idea of reverse search is to arrange all objects to be enumerated in a tree, where each search node has a unique parent node. A major task of a reverse search algorithm is the definition of a parent operation on the sets being enumerated that reduces a node in the tree to its unique parent node [25]. All the objects to be
enumerated form an enumeration tree with tree nodes representing edges and the connections between objects and the corresponding parent are represented by edges. A child operation, defined by inverting the parent operation, determines if an object is a valid child of a given parent object. The enumeration tree is constructed by applying a depth-first traversal, starting from a canonical root and employing the child operation to generate objects. In reverse search at each object, the algorithm identifies a superset of all children of the object, tests the parent operation to confirm whether a member object is a valid child of the parent object, and recursively searches each child [25]. Several reverse search-based algorithms have been proposed for solving traditional enumeration problems, including all induced connected subgraphs, all spanning trees of a graph, all topological orderings of an acyclic graph, all dense subgraphs of a graph, and all maximal independent sets of a graph [24, 25].

A reverse search algorithm, RS-MST, for enumerating all vertex-induced connected subgraphs has been introduced in [24] where the parent subgraph of a subgraph $G$ is obtained by removing the vertex with the minimum degree in the spanning tree of $G$. A subgraph resulting from extending a subgraph $G$ with a vertex $v$ is a valid child of $G$ if vertex $v$ is a vertex with the minimum degree in the subgraph formed by adding the vertex $v$ to the subgraph $G$. A similar approach can be applied for mining all edge-induced subgraphs. For an undirected graph $G(V,E)$ and a connected edge set $E_s \subset E$ inducing a subgraph $G[E_s]$, an edge $e \in E \setminus E_s$ is a valid extension of $G[E_s]$ if $e$ is an edge connecting to a leaf node in the minimum spanning tree of the subgraph formed by extending $G[E_s]$ with the edge $e$, i.e., $G[E_s \cup \{e\}]$. For these two reverse search algorithms, finding the MST to check for valid subgraph extension is a costly operation, considering that some extensions (invalid ones) will not be pursued in constructing the enumeration tree and will not be reported.

In [26], we proposed a reverse search algorithm for enumerating all vertex-induced connected subgraphs of a graph. The parent operation is based on the shortest distance of the newly added vertex to the anchor vertex of the subgraph. The algorithm outperformed existing methods for vertex-induced subgraph enumeration. Moreover, we employed the enumeration approach to mine all maximal cohesive subgraphs from vertex-attributed graphs. Pruning strategies were proposed to prune branches of the enumeration tree that will not result in maximal cohesive subgraphs. The proposed method takes an edge-growth approach to mine all connected frequent edgesets.

**Methods**

The backbone of the proposed frequent subgraphs mining algorithm is an approach to enumerate all connected edge-induced subgraphs of a single graph. We first explain our enumeration approach for all connected edge-induced subgraphs and then extend this approach to mine all frequent and maximal connected subgraphs.

**Enumerating all edge-induced Subgraphs**

Let $G = (V,E)$ be an undirected graph, where $V = \{v_1, ..., v_n\}$ denotes the set of vertices, and $E \subseteq V \times V$ is the set of edges. For a vertex $v_i \in V$, $i$ is a unique
identifier of that vertex, which is fixed but arbitrarily assigned. For an edge set \( E' \subseteq E \), let \( G[E'] = (V', E') \) denotes the subgraph of \( G \) induced by \( E' \), whose nodes, \( V' \) include all the endpoints of the edges in \( E' \). We call \( E' \) a connected edge set if its corresponding edge-induced subgraph \( G[E'] \) is connected. A connected edge-induced subgraph can be uniquely identified by its corresponding connected edge set and thus the two terms are used interchangeably. To maintain an edge ordering, an edge between vertices \( v_i \) and \( v_j \) is denoted as \((i, j)\) where \( i < j \). We define a total order relation on the set of edges in the graph such that \((i, j) \preceq (k, l)\) if \( i < k \) or \( i \) equals \( k \) and \( j \leq l \). The distance between two edges, denoted \( d(e_i, e_j) \), in a connected graph is the number of non-terminal vertices in a shortest path between the edges. Using this definition, adjacent edges that share an endpoint have a distance of 1; For an edge \( e \), the set of all adjacent edges of \( e \) is referred to as the neighborhood of \( e \), and is denoted as \( N(e) \). The neighborhood of \( e \) is defined as the set of edges with a distance of 1 to \( e \). \( N(e) = \{ e_i \in E, d(e_i, e) = 1 \} \). For an edgeset \( S \subseteq E \), the set of neighboring edges in a graph \( G(V, E) \), denoted as \( N(S) \), contains the set of edges not in \( S \) that have at least one neighboring edge in \( S \).

\[
N(S) = \{ e_i \in E \setminus S, \exists e \in S \text{ such that } d(e_i, e) = 1 \}
\]

**Problem Definition:** Given an undirected graph \( G(V, E) \), enumerate all connected edgesets, \( CEIS(G) \).

\[
CEIS(G) = \{ S \mid S \subseteq E \text{ and } G[S] \text{ is connected} \}
\]

In this paper, we propose a reverse search algorithm for enumerating all connected edge sets of an undirected graph.

**Search Graph**

For a single connected graph, the enumeration of the set of connected edgesets can be represented by a directed search graph in which nodes represent connected edgesets and there is a directed edge between two edgesets, \((X, Y)\) if \( Y = X \cup \{ e \} \) and the deletion of \( e \) from \( Y \) keeps \( X \) connected. In the search graph, a search node (say, \( Y \)) can have multiple incoming edges as multiple connected edgesets can lead to the same connected edgeset. A naive approach to traverse the entire set of all connected edgesets is to grow an edgeset by extending it with one of its neighbor edges and checking in a global list whether the edgeset has been enumerated before to avoid duplicate listings. Given the combinatorial nature of connected edgesets, this approach is inefficient as it enumerates each edgeset many times, and the number of distinct edgesets grows exponentially with the size of the graph.

**Reverse Search**

The algorithm builds and traverses the connected subgraphs search tree wherein nodes in the tree correspond to connected subgraphs and arcs correspond to the parent-children relations between these subgraphs. The arcs in the search tree are defined by a neighborhood function that defines a set of search nodes that can
be generated from a search node; this set is referred to as the valid children of a search node. Each edgeset appears only once in the search tree, and there is only one incoming link to each edgeset from its parent edgeset. We enumerate the set of connected edgesets by depth-first traversal of the search tree. In this section, we define the parent operation and a data structure that allows for efficient parent/child operations.

Parent child relationship
A search node \( Y \) corresponding to a connected edgeset can be obtained from a unique search node (say \( X \)). Then \( X \) is called the parent node and \( Y \) is called the child node. The edgeset \( X \) can be obtained by deleting a specific edge from the edgeset \( Y \).

**Lemma 0.1** For a connected edgeset \( U \), let \( s \) be the smallest edge in \( U \), denoted as \( \text{anchor}(U) \) and \( e \in U \) is one of the edges with the longest shortest path from \( s \), then \( G[U - e] \) is also connected.

**Proof** We will prove this claim by contradiction. Say, for a connected edgeset \( U \), \( e \) is the edge with the longest shortest distance from \( s \) and for contradiction, let’s assume that deleting \( e \) results in a disconnected graph. This means that there exists at least an edge \( e' \) such that all shortest paths between \( s \) and \( e' \) go through \( e \). Let \( p_{s e'} = (s, \cdots, e, \cdots, e') \) be a shortest path from \( s \) to \( e' \) and \( w(p_{s e'}) \) denote the length of the path. So, the shortest distance between \( s \) and \( e' \), \( w(p_{s e'}) + w(p_{e e'}) \), is greater than the shortest distance between \( s \) and \( e \), i.e., \( w(p_{s e'}) > w(p_{s e}) \). This contradicts our assumption that \( e \) is the edge with the longest shortest path distance from \( s \) in \( U \). Thus, \( G[U - e] \) is connected. \( \Box \)

For defining the child/parent relation as stated above, we need to designate an edge of the subgraph as the anchor edge. We denote the smallest edge \( s \in U \) as \( \text{anchor}(U) \), i.e., \( \text{anchor}(U) = s \) such that \( s \leq e_i, \forall e_i \in U \setminus s \). Moreover, let \( e = \text{utmost}(U) \) denote the edge in \( U \) with the longest distance to the anchor edge \( s = \text{anchor}(U) \). If there are more than one edge whose distance equals the longest distance, we take the smallest edge according to the order relation. For a connected edgeset \( U \) with anchor \( s \), the utmost edge is defined as follows: \( \text{utmost}(U) = e \) such that \( e \in U \setminus s \) and \( \forall e_i \in U \setminus e \) either \( d(e_i, s) < d(e, s) \) or \( d(e_i, s) = d(e, s) \) and \( e_i \preceq e \).

**Valid Children**

Building on lemma 0.1, in the search tree, we can expand a parent node \( X \) to construct one of its child node \( Y \). In a given graph \( G(V, E) \), the search tree node \( X \) corresponds to the connected edgeset \( U \) with \( s = \text{anchor}(U), e = \text{utmost}(U) \). Now, \( e' \in E \setminus U \) is a neighbor edge of \( U \), and also \( s \leq e' \). Then, \( Y \) is a child of \( X \) corresponding to the connected edgeset, \( U' = U \cup \{e'\} \) if and only if the following condition holds:

1. The distance from \( s \) to \( e' \) is greater than the distance from \( s \) to \( e \), or
Both \( e' \) and \( e \) have the same distance to \( s \), but \( e \preceq e' \).

This above definition of \textit{validchild} ensures that the newly added edge to the child node has the longest distance from the anchor, and if multiple edges have the same longest distance to the anchor, the newly added edge is the largest considering the order relation among the edges. The proposed reverse search parent-child relation is the backbone of the enumeration tree neighborhood function, \( \mathcal{N} : \text{CEIS}(G) \rightarrow 2^{\text{CEIS}(G)} \). The parent-child relation guarantees that a search node only appears once in the range of the neighborhood function.

If \( U \) and \( U \cup \{e'\} \) are edgesets corresponding to a parent and a child node, respectively, we call \( e' \) a valid candidate of \( U \), otherwise, we call it an invalid candidate of \( U \). For an edgeset \( U \), the set of neighboring edges constitute the candidate edges (valid and invalid). Figure 1 (a) shows a sample graph, and figure 1 (b) shows the enumeration tree of the set of all connected edgesets of this graph. Edges are uniquely labeled starting from 1. The edges of an edgeset are written inside the oval shape and the set of candidate edges are written adjacent to the oval shape. Figure 1 (b) shows that edgeset \( U = \{1,3\} \) has \( \{2,4,5\} \) as the candidate edges; \( \text{anchor}(U) = 1 \) and \( \text{utmost}(U) = 3 \). Edge 2 is not a valid candidate because its distance to 1 is the same as the distance of the utmost edge 3 to edge 1 but 2 is less than 3 in the order of the edges, thus the branch corresponding to \( \{1,3,2\} \) will not be explored. Edge 4 has the same distance as edge 3 to edge 1, but since 4 is greater than 3, then edge 4 is a valid candidate. Edge 5 distance to edge 1 is larger than the distance of the utmost edge and thus it is a valid candidate. For the edgeset \( \{2,5\} \) with a candidate set \( \{1,3,4\} \), both edges 1 and 3 are not valid candidates; edge 1 is less than the anchor 2 and edge 3 has the same distance as edge 5 to edge 2 but edge 3 is less than 5; edge 4 is a valid candidate because its distance to edge 2 is larger than the distance of the utmost edge 5. For single-edge search nodes in level 1, if the candidate edge is larger than the anchor edge, then it is an invalid edge.
Enumerating all subgraphs of a single graph

Algorithm 1 shows the pseudo-code for our algorithm. For each edge in the graph, we call EnumerateCEIS, a recursive procedure; the set of neighbors for each edge constitute the candidate edges. The procedure takes a connected edgeset $U$, the set of candidate edges, and the last edge added to the edgeset. For each edge $e_j$ in the candidate set (line 7), the procedure checks if the edge is a valid candidate (line 8) for extending $U$. If so, it updates the candidate set and recursively calls the EnumerateCEIS procedure (lines 9-10). The candidate set can be updated by using the current candidate set and the neighbors of the last added edge $N(e_j)$. The isValidExtension procedure (line 15) checks if the edge $e_j$ is a valid candidate for the edgeset in $E_s$ following the rules in the valid children section.

Complexity Analysis

Since the number of reported patterns can be exponential in the number of edges of the graph, we analyze the time the algorithm takes to report a pattern after it has generated the previous pattern. For a polynomial delay algorithm, it takes polynomial time to output an element after generating the previous element [24].

We use an array-based implementation in which we maintain the set of edges of an edgeset, the candidate edges and the distance of the candidate edges to the anchor edge. Using this data structure, the anchor edge, utmost edges, and the distance of an edge to the anchor edge can be accessed in constant time. The algorithm checks if an edge is a valid candidate of the edgeset in a constant time $O(1)$ (Algorithm 1 line 8). If the edge is a valid candidate, then the candidate set can be updated in $O(|N(e_j)|)$ time for the last added edge $e_j$. Once a valid candidate is encountered, the recursive procedure is called. Therefore, each connected edgeset can be enumerated with linear delay $O(|E|)$.

An algorithm is output polynomial, if it outputs all the elements to be enumerated in time polynomial to the number of elements. Since the proposed algorithm takes linear time for each connected edgeset, it is output (or total) polynomial in the number of connected edgesets.

The algorithm explores the search tree in a depth first manner, which ensures that the space used is bounded by the depth of the search tree, which is at most $|E|$. We use three arrays, each of size $|E|$ to keep track of which edges are in the connected edgeset, their neighbors, and their distances to the anchor edge. So, the depth first search of the enumeration tree can be done with linear space in the depth of the enumeration tree which is $O(|E|)$.

Mining Frequent Subgraphs

In many applications, we have a dataset of graphs and the goal is to extract significant subgraphs. In the frequent subgraph mining problem, the goal is to mine subgraphs that appear in at least a user-defined minimum threshold of the graphs. In this work, we are only concerned with connected frequent subgraphs.

Graph Dataset Let $\mathcal{G} = \{G_1, G_2, \cdots, G_n\}$ denote a set of $n$ undirected graphs. An undirected graph $G_i = (V, E_i)$ is a tuple where $V = \{v_1, v_2, \cdots, v_k\}$ denote the set of vertices, and $E_i \subseteq V \times V$ denote the set of edges of the corresponding graphs. All the graphs are defined over the same set of vertices.
Algorithm 1 Mining All Connected Edge-Induced Subgraphs

**Input:** $G = (V, E)$: an undirected graph

1: for $e_i \in E$ do
2:     ENUMERATECEIS($\{e_i\}$, $\text{Neighbors}(\{e_i\}), e_i$)
3: end for

5: function ENUMERATECEIS($E_s$, $C$, $e_l$)
6:     output $E_s$
7:     for $e_j \in C$ do
8:         if isValidExtension($E_s$, $e_j$, $e_l$) then
9:             $C' = \text{Neighbors}(E_s \cup \{e_j\})$
10:                ENUMERATECEIS($E_s \cup \{e_j\}$, $C'$, $e_j$)
11:         end if
12:     end for
13: end function

15: function isValidExtension($E_s$, $e_j$, $x$)
16:     $s = \text{anchor}(E_s)$
17:     if $e_j < s$ then
18:         return False
19:     end if
20:     if $\text{distance}(e_j, s) > \text{distance}(x, s)$ then
21:         return True
22:     end if
23:     return $\text{distance}(e_j, s) = \text{distance}(x, s)$ and $e_j > x$
24: end function

In this work, we represent the dataset $\mathcal{G}$ of $n$ graphs as an edge-attributed graph, $G(V, E, f)$, where $V$ is the set of vertices and $E$ is the set of all the edges in the graph dataset and an edge attribute function. The edge attribute function maps each edge to the set of graphs in which it appears. The set of all edges is the union of the sets of edges in each graph.

$$E(\mathcal{G}) = \bigcup_{i=1}^{n} E_i,$$

where $E_i$ is the set of edges in $G_i \in \mathcal{G}$. We label the edges in the edge-attributed graph with unique identifiers $\{1, 2, \cdots, |E|\}$. Figure 2 shows a toy graph dataset of four graphs in (a) and the corresponding edge-attributed graph in (b).

**Subgraph** A graph $G_s = (V_s, E_s)$ is a subgraph of $G_i = (V, E_i)$, denoted as $G_s \subseteq G_i$, if and only if $V_s \subseteq V$ and $E_s \subseteq E_i$.

**Edge-Induced Subgraph** For a graph $G(V, E)$, and an edgeset $S \subseteq E$, the edge-induced subgraph, denoted as $G[S]$, is a subgraph $G[S] = (V_S, S)$ whose edge set is $S$ and the vertex set $V_S$ includes all the endpoints of edges in $S$.

**Supporting Graphs** Given a set of graphs $\mathcal{G}$, the set of supporting graphs of an edge-induced subgraph, $G_s$, is defined as follow: $\text{sup}(\mathcal{G}, G_s) = \{G_i | G_s \subseteq G_i and G_i \in \mathcal{G}\}$.
Mining Frequent Subgraphs

Frequent Subgraph Given a graph dataset \( \mathcal{G} \) and user-specified support threshold \( S_{\text{min}} \), a graph \( G_s \) is called frequent if the subgraph’s support is equals to or greater than the support threshold, i.e., \( |\text{sup}(G_s)| \geq S_{\text{min}} \).

Since an edge-induced subgraph is uniquely identified by the edgesset, we use frequent subgraphs and frequent edgessets interchangeably.

Problem Definition: Given a graph dataset \( \mathcal{G} \) and a support threshold \( S_{\text{min}} \), the problem of mining the set of frequent subgraph is to enumerate the set:

\[
\mathcal{F} = \{ G_{s1}, G_{s2}, G_{s3}, \ldots, G_{s|\mathcal{F}|} \}
\]

such that every \( G_{si} \in \mathcal{F} \) is a frequent connected subgraph, i.e., \( |\text{sup}(G_{si})| \geq S_{\text{min}} \).

For the graph dataset in Figure 2(a), the set of frequent subgraph for minimum support of 3 is shown in Figure 2(c). Given a minimum support threshold \( S_{\text{min}} \), the anti-monotone support constraint guarantees that if a subgraph \( G_s \) is frequent, then each subgraph of the subgraph is also frequent, i.e., \( |\text{sup}(G_s)| \geq S_{\text{min}} \Rightarrow \) for all \( G^* \subset G_s \), the subgraph is frequent \( |\text{sup}(G^*)| \geq S_{\text{min}} \).

Our proposed algorithm for mining all frequent subgraphs problem employs the reverse search enumeration approach in algorithm 1 to enumerate all connected subgraph and enforcing the supporting constraint. The anti-monotone property of the support of a subgraph is employed in the mining algorithm to prune search branches when an infrequent subgraph is encountered. If an infrequent subgraph is encountered, then the recursion procedure EnumerateFCIS is not called and the search
The algorithm for mining frequent subgraphs is shown in Algorithm 2. In line 1, infrequent edges are pruned, and the recursive EnumerateFCIS procedure is called for each frequent edge (Line 3). The recursive procedure follows the same steps as the enumeration approach in Algorithm 1, except for the if statement in line 9 to ensure that only search branches rooted at frequent subgraphs are explored. The recursive procedure is called only from frequent children (line 11). Therefore, only frequent subgraphs will be added to the set of frequent subgraphs in line 6.

Algorithm 2 Mining All Connected Frequent Subgraphs

Input: a graph dataset, \( G \), and a minimum support threshold, \( S_{\min} \)
Output: \( F \), all frequent subgraphs

1: \( L_1 = \{ e \mid e \in E \text{ and } |sup(e)| \geq S_{\min} \} \)
2: for \( e \in L_1 \) do
3:     EnumerateFCIS(\( \{ e \} \), Neighbors(\( \{ e \} \)), \( e \))
4: end for
5: function EnumerateFCIS(\( E_s \), \( C \), \( e_l \))
6:     \( F \leftarrow F \cup E_s \)
7:     for \( e \in C \) do
8:         if isValidExtension(\( E_s \), \( e \), \( e_l \)) then
9:             if \( |sup(E_s)| \geq \delta \) then
10:                \( C' = \text{Neighbors}(E_s \cup \{ e \}) \)
11:                EnumerateFCIS(\( E_s \cup \{ e \} \), \( C' \), \( e_l \))
12:         end if
13:     end if
14: end for
15: end function
16: return \( F \)

Problem Definition: Given a graph dataset \( G \) and a support threshold \( S_{\min} \), the problem of mining the set of maximal frequent subgraph is to enumerate the
Figure 3 Mining Frequent and Maximal Subgraphs: (a) A graph dataset of four graphs in the edge-attributed graph format (b) Enumeration tree for frequent subgraph with minimum support set to 3; maximal frequent subgraphs are circled with dotted line; all the subtrees with the light gray background are pruned when mining for only maximal frequent subgraphs in the proposed algorithm.

Figure 3

\[ M = \{G_{s1}, G_{s2}, G_{s3}, \ldots, G_{s|M|}\} \]

such that every \( G_{si} \in M \) is a maximal frequent connected subgraph.

For the graph dataset of four graphs shown in figure 2(a), and minimum support \( S_{min} = 3 \), there are two maximal frequent subgraphs and they are drawn inside dotted circle in figure 2(c).

In the enumeration tree for mining frequent subgraphs, every leaf search node is potentially a maximal frequent subgraph. The reason for a leaf not being a maximal frequent subgraph is that there could be an invalid subgraph of that leaf that is frequent and it was not explored because it is not a valid extension at this stage of the enumeration tree. An algorithm for mining all maximal frequent subgraphs is to enumerate the frequent subgraphs enumeration tree and to report subgraphs that do not have any frequent valid or invalid extension. This algorithm is a straightforward extension of Algorithm 2. Following this mining approach, the enumeration tree for maximal frequent subgraphs would look like the tree in Figure 3(b). We will need to enumerate all 20 frequent subgraphs to get the two maximal subgraphs. Enumerating the search tree of frequent subgraphs is computationally expensive, especially for low minimum support thresholds when the search tree becomes very large. A more efficient approach would be to mine the set of all maximal frequent subgraphs without enumerating the whole frequent subgraphs enumeration tree. In the following subsections, we develop pruning strategies that eliminate the need to traverse search branches without sacrificing the completeness of the results. In the experiments section, we demonstrate how the proposed pruning strategies result in a significant performance improvement.
**Consumed by a sibling**

In a given graph \(G(V, E)\) and a connected frequent edgeset \(S \subseteq E\) and the corresponding edge-induced frequent subgraph \(G[S]\), let \(e_i\) and \(e_j\) be two valid candidate edges of \(G[S]\) such that \(e_i\) is closer to anchor\((S)\) than \(e_j\) \((e_i <_S e_j)\), and these two extensions generate two frequent subgraphs, \(G[S \cup \{e_i\}]\) and \(G[S \cup \{e_j\}]\) and the set of supporting graphs of \(G[S \cup \{e_i\}]\) is a subset of the supporting graphs of \(G[S \cup \{e_j\}]\), i.e., \(sup(G[S \cup \{e_i\}]) \subseteq sup(G[S \cup \{e_j\}])\). Under this scenario, these two subgraphs are not maximal frequent subgraphs because any maximal frequent subgraph that is a supergraph of \(G[S \cup \{e_i\}]\) will also be a supergraph of \(G[S \cup \{e_j\}]\). This conclusion is reached by observing that any maximal subgraph that is a supergraph of \(G[S \cup \{e_i\}]\) will be explored through the \(G[S \cup \{e_i\}]\) branch. In this case, we can safely prune the search branch rooted at \(G[S \cup \{e_j\}]\).

**Lemma 0.2** For an edge-induced connected frequent subgraph, \(G[S]\), and two valid candidates \(e_i\) and \(e_j\) closer to anchor\((S)\) than \(e_j\) and two frequent children subgraphs \(G[S \cup \{e_i\}]\) and \(G[S \cup \{e_j\}]\) with \(sup(G[S \cup \{e_j\}]) \subseteq sup(G[S \cup \{e_i\}])\), the search branch rooted at \(G[S \cup \{e_j\}]\) can be safely pruned.

**Proof** At the search node in the enumeration tree representing \(G[S]\), consider the set of all remaining neighboring edges \(X \subseteq E \setminus \{S \cup \{e_i, e_j\}\}\), and assume there is a maximal subgraph in the branch rooted at \(G[S \cup \{e_j\}]\) which we plan to prune. Such a maximal frequent subgraph is denoted as \(G' = G[S \cup \{e_j\}] \cup Y\). However, this \(G'\) subgraph can be extended with \(e_i\) and still yield a frequent subgraph. The subgraph \(G[S \cup \{e_j\}] \cup Y \cup \{e_i\}\) is a frequent subgraph since \(e_i\) is connected to \(S\) and can be added to \(G[S \cup \{e_j\}] \cup Y\) without violating the minimum support threshold since \(sup(G[S \cup \{e_j\}]) \subseteq sup(G[S \cup \{e_i\}])\). The existence of a frequent subgraph \(G[S \cup \{e_j\}] \cup Y \cup \{e_i\}\) contradicts our assumption that \(G[S \cup \{e_j\}] \cup Y\) is a maximal frequent subgraph. This proves that \(G[S \cup \{e_i\}] \cup Y\) is not a maximal frequent subgraph.

Moreover, \(G[S \cup \{e_j\}] \cup Y \cup \{e_i\}\) is not a descendant of \(G[S \cup \{e_j\}]\) since \(e_i\) is not a valid extension to any descendant of \(G[S \cup \{e_j\}]\). This is because once \(e_j\) is added to the edge set \(S\), edge \(e_i\) can not be added since \(e_i\) is closer to anchor\((S)\) than \(e_j\) and according to the valid children definition, \(e_i\) will never be a valid extension of any descendant of \(G[S \cup \{e_j\}]\). Since all the subgraphs in the search tree rooted at \(G[S \cup \{e_i\}]\) can be expressed as \(G[S \cup \{e_j\}] \cup Y\), for \(Y \subseteq E \setminus \{S \cup e_i, e_j\}\), this proves that none of the descendants of \(G[S \cup \{e_j\}]\) will be a maximal frequent subgraph. Therefore, it is safe to prune the search branch rooted at \(G[S \cup \{e_j\}]\) without losing any maximal frequent subgraphs. \(\square\)

For the graph dataset in Figure 3, and minimum support \(S_{\min} = 3\), consider the edgeset \(\{(A, B)\}\) and its two neighboring edges \((A, C)\) and \((B, C)\). The supporting
graphs of the subgraph induced by the edgeset \{(A, B), (A, C)\} are graphs \{1, 2, 3\} which is the same as the supporting graphs for the subgraph induced by the edgeset \{(A, B), (B, C)\}. Moreover, since edge \((A, C)\) is closer than \((B, C)\) to the anchor edge \((A, B)\), then by applying lemma 0.2, the search tree rooted at \((A, B)\) can be safely pruned. Similarly, the search branch at the subgraph induced by \{(A, B), (B, D)\} is pruned because it is subsumed by the search branch at \{(A, B), (A, D)\}.

**Identical as Parent**

Employing lemma 0.2, we devise another pruning strategy. For a given search node representing \(G[S]\), we sort all valid candidates with respect to \(S\), if we encounter an extension \(e_i\) such that the new extended subgraph appears in the same set of graphs as the parent subgraph, then all the remaining valid candidates in the sorted list can be safely skipped. Assume that there is an edge \(e_j\) that is a valid extension of \(G[S]\) and \(e_j\) is closer to \(anchor(S)\) than \(e_j\). The supporting graphs of \(G[S \cup e_j]\) is a subset of the supporting graphs of \(G[S]\) \((sup(G[S \cup e_j]) \subseteq sup(G[S]))\) because the set of the supporting graphs of a subgraph is a subset of the set of the supporting graph of the parent subgraph. This pruning strategy an an extension of lemma 0.2. Since we have \(sup(G[S \cup e_j])\) equals \(sup(G[S])\) and therefore \(sup(G[S \cup e_j]) \subseteq sup(G[S \cup e_i])\), then by lemma 0.2, the search branch rooted at \(e_j\) can be safely pruned.

For the example in Figure 3, the supporting graphs of the subgraph induced by the edgeset \{(A, B), (A, C), (B, C)\} are the same as the supporting graphs of the parent subgraph \{(A, B), (A, C)\}, therefore all the remaining search branches for the parent graph can be safely pruned. This is the case for the subgraph induced by edgeset \{(A, B), (A, C), (C, D)\}.

**Level One Pruning**

Pruning for level one (single edge) is similar to pruning at any search node in the search tree. For any two edgesets, \(e_i\) and \(e_j\) sharing a common endpoint and \(e_i\) is smaller than \(e_j\) in the order of edges, if the supporting graph of \(e_j\) is a subset of the supporting graphs of \(e_i\) \((sup(e_j) \subseteq sup(e_i))\), then the search tree rooted at \(e_j\) can be safely pruned. The proof follows the same steps as in lemma 0.2. For an edgeset \(S \subseteq E \setminus \{e_i, e_j\}\), the subgraph \(G[e_j \cup S]\) is not a maximal frequent subgraph since \(G[e_j \cup S \cup e_i]\) is a frequent subgraph. Moreover, \(G[e_j \cup S \cup e_i]\) is not a descendant of \(G[e_j]\) since \(e_i < e_j\).

In Figure 3, the search subtree rooted at \((A, C)\) is pruned because the set of supporting graphs of \((A, B)\) is a superset of the supporting graphs of \((A, C)\). Similarly, the three search subtrees rooted at \((A, D)\), \((B, C)\), and \((B, D)\) are all safely pruned.

**Algorithm**

Algorithm 3 shows the pseudo code for the proposed algorithm. The algorithm follows the enumeration approach for mining frequent subgraphs and employs the pruning strategies to avoid visiting subtree branches that will not result in maximal frequent subgraphs. In line 1, frequent edges are extracted and then in lines 3-7, a search subtree will be expanded from each frequent edge. Frequent edges that are covered by a neighboring smaller edges will not be explored (line 4). The neighbors...
of edge $e$ constitute the candidate edges. In the MineMaximalSubgraph procedure, for each edge in the candidate edges $C$, if the extension would generate a frequent subgraph, then we set the maximal flag to false indicating that the current subgraph is not a maximal frequent subgraph. Next, for each valid candidate we check if this extension is covered by a previous extension according to the two pruning strategies and we recursively call the procedure only for valid children that are not covered (lines 14-19). We add the current pattern to the maximal frequent subgraphs set if the maximal flag is still true, line 24.

**Algorithm 3** MAFSA: mining all MAximal Frequent SubgrAphs

**Input:** a graph dataset, $G$, and a minimum support threshold, $S_{\text{min}}$

**Output:** $M$, all maximal frequent connected subgraphs (edgesets)

1: $L_1 = \{e \mid e \in E \text{ and } \text{sup}(e) \geq S_{\text{min}}\}$

2: $M = \{\}$

3: for $e \in L_1$ do

4: if $e$ is not covered by a smaller neighboring edge then

5: MineMaximalSubgraphs($\{e\}$, $N(\{e\})$, $e$)

6: end if

7: end for

8: function MineMaximalSubgraphs($E_s$, $C$, $e_l$)

9: maximal $\leftarrow$ true

10: for $e_i \in C$ do

11: $E'_s \leftarrow E_s \cup e_i$

12: if $|A(E'_s)| \geq S_{\text{min}}$ then

13: maximal $\leftarrow$ false

14: if isValidExtension($E_s$, $e_i$, $e_l$) then

15: if $e_i$ covered by a smaller edge in $C$ then

16: Continue

17: end if

18: $C' = \text{Neighbors}(E_s \cup \{e_i\})$

19: MineMaximalSubgraphs($E'_s$, $C'$, $e_l$)

20: end if

21: end if

22: end for

23: if maximal then

24: $M = M \cup E_s$

25: end if

26: end function

27: return $M$

**Results**

We tested the performance of the proposed algorithm on mining frequent and maximal subgraphs from gene coexpression networks. Moreover, for investigating the impact of the pruning techniques, we compared the running time of the algorithm with and without the pruning techniques. All experiments were performed on a machine with Intel Xeon 2.40GHz processor with 16 Gbytes main memory, running the Linux operating system. The algorithms were implemented in C++ and the MULE implementation was in C.

0.1 Performance on real data

We tested the proposed algorithm on 35 tissue gene coexpression networks constructed by the Gene Genetic Network Analysis Tool [27]. The coexpression net-
works were inferred from Genotype-Tissue Expression (GTEx) data. Each co-expression network is constructed from the gene expression of non-diseased tissue samples. On average there are 14,415 coexpression links (edges) in each network among 9,998 genes. In total, there are 1,548,622 unique coexpression edges that appear in at least one coexpression network. Among these edges, there are 4,127 edges that appear in at least 20 networks, and on average each edge appears in 3.28 networks.

Table 1 shows how the number of frequent and maximal subgraphs ($|F|$ and $|M|$) and the running times for the MULE and RASMA for mining the frequent and maximal subgraphs vary for varying minimum support thresholds. For mining the maximal subgraphs, the proposed algorithm is orders of magnitude faster than the MULE algorithm for low support thresholds. The MULE algorithm is much slower for mining maximal frequent subgraphs since it has to enumerate the same frequent subgraphs enumeration tree. Moreover, for each potential maximal subgraph the MULE algorithm checks if it has a supergraph in a global list. For mining all the frequent subgraphs, both algorithms have similar running times and for a support threshold of 15 both did not finish the mining task in two days.

Table 2 shows the topological properties of the reported subgraphs and running times of RASMA for lower support thresholds. The number of maximal subgraphs ($|M^*|$) increases for lower support and so do the average numbers of edges ($|E|$), nodes ($|V|$), and density ($Density$). For calculating the topological properties of the maximal frequent subgraphs, only subgraphs with at least three edges (denoted $|M^*|$) are considered since a large percentage of the maximal frequent subgraphs have one or two edges only.

**Effectiveness of Pruning Techniques**

The pruning techniques aim at reducing the number of search nodes explored to report the maximal patterns mining. The closer to the root the pruning occurs, the more search nodes are eliminated. To investigate how the proposed pruning techniques improve the performance of the algorithm, we show a comparison with the pruning techniques disabled. Table 4 shows the impact of the pruning techniques on the running time and on the number of frequent subgraphs explored. Although both the frequent subgraphs and maximal subgraphs (without pruning) algorithms enumerate the same frequent subgraphs search tree, it is important to notice that the algorithm without pruning is much slower than mining all frequent subgraphs. This is true since for a maximal subgraph, all immediate potential children nodes have to be check for frequency to mark the subgraph as maximal (line 12 in algorithm 3), regardless of whether the extension is a valid child. Therefore, the number of frequency checking is much larger than the number of frequent search nodes explored. However, the algorithm for frequent subgraphs checks if an extension is frequent only for the valid children (lines 8–9 in algorithm 2). The algorithm with pruning strategies traverses only a very small fraction (0.000028 for support = 16) of the frequent subgraphs while maintaining the completeness of the maximal frequent subgraphs.

[1] https://www.gtexportal.org/
Analysis of Maximal Frequent Subgraphs

We performed a biological enrichment for the gene sets (nodes) of the maximal frequent subgraphs. A biological annotation, knowledge, is to said to be enriched in a gene set if a significant subset of the genes of the gene set are annotated with the given annotation. We tested the overrepresentation of genes with a specific annotation in a gene set using the hypergeometric test (with $p_{value} = 0.01$). We used multiple annotation databases from the Molecular Signatures Database (MSigDB) [28, 29] for assessing the enrichment of the genes in these reported patterns with these annotations.

1 KEGG: Gene sets derived from the KEGG pathway database (186 sets).
2 BP: Gene sets derived from the GO Biological Process Ontology. (7350 sets).
3 MF: Gene sets derived from the GO Molecular Function Ontology. (1645 sets).

Table 3 shows the percentage of subgraphs that are biologically enriched for each of the three biological signatures. Some patterns are enriched with several signatures and some signatures are enriched in the genes of multiple subgraphs. Moreover, the reported subgraphs are enriched with a large number of biological annotations for each of the databases. Only maximal frequent subgraphs with at least three edges were considered for the analysis (denoted as $|M^*|$ in the table). The enrichment analysis shows that frequent subgraphs are highly enriched with known biological annotations.

An annotation can be enriched in many reported gene sets. We sorted the annotations by the number of subgraphs they are enriched in. Table 5 shows the top biological signatures that were enriched the most in the reported patterns for $S_{min} = 20$.

Conclusion

Frequent coexpression subnetworks have been shown to be effective in functional annotation and subnetwork biomarker discovery. We proposed a reverse search algorithm for mining maximal frequent subgraphs. We first proposed a reverse search strategy for enumerating all edge-induced subgraphs from a single graph. The enumeration approach is then employed for mining frequent and maximal subgraphs. To eliminate search branches that will not result in maximal frequent patterns, we proposed pruning strategies that employ the order in which branches are enumerated. The pruning strategies are possible because the reverse search enforces strict definition on the order in which search nodes are enumerated. Experiments on gene coexpression datasets demonstrate the effectiveness of the proposed approaches. The proposed approach is thousands of times faster than the existing algorithm. Enrichment analysis of the genesets in the maximal frequent subgraphs reveal that maximal frequent coexpression subnetworks are enriched with known biological annotations.

Competing interests

The authors declare that they have no competing interests.

Author’s contributions

SS and MA discussed the idea and designed the algorithm. SS worked on experimental design and patterns topological analysis. SS, MA and MAH wrote the paper. MA contributed to this work when he was a graduate student at North Dakota State University. All the authors read and approved the final manuscript.
Declarations
Not applicable

Ethics approval and consent to participate
Not applicable

Consent for publication
Not applicable

Availability of data and material
The dataset and the implementation of the algorithm is available at http://www.cs.ndsu.nodak.edu/~ssalem/multirelation.html.

Funding
Acknowledgments
The authors would like to acknowledge the authors of the MULE algorithm for making the software available and for giving feedback on how to extend the MULE algorithm for mining frequent and maximal subgraphs.

Author details
1 North Dakota State University, Fargo, ND, 58102 USA. 2 Indiana University–Purdue University Indianapolis, Indianapolis, IN, 46202 USA.

References
1. Lee, H.K., Hsu, A.K., Sajdak, J., Qin, J., Pavlidis, P.: Coexpression analysis of human genes across many microarray data sets. Genome Res. 14(6), 1085–1094 (2004)
2. Hu, H., Yan, X., Huang, Y., Zhou, X.J.: Mining coherent dense subgraphs across massive biological networks for functional discovery. Bioinformatics 21 Suppl 1, 213–221 (2005)
3. Li, W., Liu, C.-C., Zhang, T., Li, H., Waterman, M.S., Zhou, X.J.: Integrative analysis of many weighted co-expression networks using tensor computation. PLoS Comput Biol 7(6), 1001106 (2011)
4. Iedeker, T., Ozier, O., Schwikowski, B., Siegel, A.F.: Discovering regulatory and signalling circuits in molecular interaction networks. Bioinformatics 18(Suppl 1), 233–40 (2002)
5. Chuang, H., Lee, E., Liu, Y., Lee, D., Ideker, T.: Network-based classification of breast cancer metastasis. Mol Syst Biol. 3, 140 (2007)
6. Li, J., Zhou, D., Qiu, W., Shi, Y., Yang, J.-J., Chen, S., Wang, Q., Pan, H.: Application of weighted gene co-expression network analysis for data from paired design. Scientific Reports 8(622) (2018)
7. Huang, Y., Li, H., Hu, H., Yan, X., Waterman, M.S., Huang, H., Zhou, X.J.: Systematic discovery of functional modules and context-specific functional annotation of human genome. Bioinformatics 23(13), 222–229 (2007)
8. Salem, S., Ozczaglar, C.: MFMS: Maximal frequent module set mining from multiple human gene expression data sets. In: Proceedings of the 12th International Workshop on Data Mining in Bioinformatics. BioKDD ’13, pp. 51–57. ACM, New York, NY, USA (2013)
9. Salem, S., Ozczaglar, C.: Hybrid co-expression link similarity graph clustering for mining biological modules from multiple gene expression datasets. BioData Mining 7(1), 16 (2014)
10. Salem, S.: Template edge similarity graph clustering for mining multiple gene expression datasets. International Journal of Data Mining and Bioinformatics 18(1), 28–39 (2017)
11. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: Proceedings of International Conference on Data Mining, ICDM ’02, pp. 313–320 (2001)
12. Yan, X., Han, J.: gSpan: graph-based substructure pattern mining. In: Proceedings of International Conference on Data Mining. ICDM ’02 (2002)
13. Kuramochi, M., Karypis, G.: An efficient algorithm for discovering frequent subgraphs. IEEE Trans. Knowl. Data Eng. 16(9), 1038–1051 (2004)
14. Nijssen, S., Kok, J.N.: The gaston tool for frequent subgraph mining. Proceedings of the International Workshop on Graph-Based Tools 127(1), 77–87 (2005)
15. Koyutürk, M., Grana, A., Szpankowski, W.: An efficient algorithm for detecting frequent subgraphs in biological networks. Bioinformatics 20(Suppl 1), 200–207 (2004)
16. Yan, X., Han, J.: CloseGraph: mining closed frequent graph patterns. In: Proceedings of International Conference on Knowledge Discovery in Data Mining. KDD ’03 (2003)
17. Huan, J., Wang, W., Prins, J., Yang, J.: SPIN: Mining maximal frequent subgraphs from graph databases. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD ’04, pp. 581–586 (2004)
18. Al Hasan, M., Zaki, M.J.: Output space sampling for graph patterns. Proc. VLDB Endow. 2(1), 730–741 (2009)
19. Yan, X., Zhou, X.J., Han, J.: Mining closed relational graphs with connectivity constraints. In: Proceedings of International Conference on Knowledge Discovery in Data Mining. KDD ’05, pp. 324–333. ACM, New York, NY, USA (2005)
20. Pei, J., Jiang, D., Zhang, A.: On mining cross-graph quasi-cliques. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. KDD ’05, pp. 228–238. ACM, New York, NY, USA (2005)
21. Jiang, D., Pei, J.: Mining frequent cross-graph quasi-cliques. ACM Trans. Knowl. Discov. Data 2(4), 16–11642 (2009)
22. Koyutürk, M., Kim, Y., Subramaniam, S., Szpankowski, W., Grana, A.: Detecting conserved interaction patterns in biological networks. Journal of Computational Biology 13(7), 1299–1322 (2006)
23. Uno, T.: Constant time enumeration by amortization. In: Algorithms and Data Structures, pp. 593–605. Springer, ??? (2015)
24. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Applied Mathematics 65(1), 21–46 (1996)
25. Eppstein, D.: All maximal independent sets and dynamic dominance for sparse graphs. ACM Trans. Algorithms 5(4) (2009). doi:10.1145/1597036.1597042
26. Alokshyia, M., Salem, S., Abed, F.: A linear delay algorithm for enumerating all connected induced subgraphs. Bioinformatics 20(12), 319 (2019)
27. Pierson, E., the GTEx Consortium, Koller, D., Battle, A., Mostafavi, S.: Sharing and specificity of co-expression networks across 35 human tissues. PLOS Computational Biology 11(5), 1–19 (2015). doi:10.1371/journal.pcbi.1004220
28. Subramanian, A., Tamayo, P., Moohra, V.K., Mukherjee, S., Ebert, B.L., Gillette, M.A., Paulovich, A., Pomeroy, S.L., Golub, T.R., Lander, E.S., Mesirov, J.P.: Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. Proceedings of the National Academy of Sciences 102(43), 15545–15550 (2005)
29. Liberzon, A., Birger, C., Thorvaldsdóttir, H., Ghandi, M., Mesirov, J., Tamayo, P.: The molecular signatures database hallmark gene set collection. Cell Systems 1(6), 417–425 (2015)

Figures

Tables

| Table 1 | Running times (in seconds) of the MULE and MAFSA algorithms. |
|---------|-------------------------------------------------------------|
| sup     |          | Frequent Patterns | Maximal Patterns |
|         | $S_{min}$ | $|F|$ | MULE | MAFSA | $|M|$ | MULE | MAFSA |
| 20      | 214,119   | 0.93 | 0.46 | 4,701 | 5.25 | 0.34 |
| 19      | 599,798   | 1.02 | 0.65 | 6,463 | 12.32 | 0.35 |
| 18      | 1,776,157 | 2.01 | 1.60 | 9,038 | 50.75 | 0.45 |
| 17      | 9,040,719 | 7.79 | 7.50 | 12,879 | 241.19 | 0.59 |
| 16      | 2,968,677,772 | 2880 | 1718 | 18,978 | 1877.95 | 0.92 |
| 15      | -         | -   | -   | 28,578 | -   | 3.50 |

| Table 2 | Topological properties and running times for varying support thresholds |
|---------|-------------------------------------------------------------|
|         | $S_{min}$ | $|M|$ | $|M^*|$ | $|E|$ | $|V|$ | Density | Seconds |
| 15      | 28,578    | 17,855 | 14.78 | 15.36 | 0.2  | 3.50 |
| 14      | 43,773    | 30,094 | 16.33 | 16.82 | 0.19 | 4.20 |
| 13      | 67,978    | 50,227 | 18.17 | 18.53 | 0.17 | 7.65 |
| 12      | 105,950   | 82,831 | 20.22 | 20.42 | 0.16 | 15.75 |
| 11      | 164,613   | 134,245 | 22.42 | 22.41 | 0.15 | 37.53 |
| 10      | 252,345   | 211,678 | 24.75 | 24.47 | 0.15 | 92.38 |
| 9       | 378,437   | 322,381 | 27.1  | 26.45 | 0.15 | 241.65 |
| 8       | 549,709   | 470,572 | 29.03 | 27.93 | 0.16 | 669.25 |
| 7       | 774,221   | 659,303 | 29.36 | 27.82 | 0.18 | 1958.77 |

| Table 3 | Enrichment analysis of the maximal frequent subgraphs |
|---------|-----------------------------------------------------|
| sup     | $|M^*|$ | $|E|$ | Density | $E_{KEGG}$ | $E_{BP}$ | $E_{MF}$ |
| 20      | 1,701   | 9.82 | 0.29 | 76.3 | 91.5 | 86.5 |
| 19      | 2,580   | 10.76 | 0.26 | 77.2 | 91.9 | 87.1 |
| 18      | 4,116   | 11.68 | 0.25 | 77.8 | 91.8 | 87.5 |
| 17      | 6,598   | 12.72 | 0.23 | 79.2 | 91.7 | 87.7 |
| 16      | 10,716  | 14.04 | 0.21 | 81.0 | 91.8 | 88.7 |
### Table 4 Impact of the pruning strategies on the number of search nodes explored.

| sup | maximal | Without Pruning | MAFSA |
|-----|---------|-----------------|-------|
| $S_{min}$ | $|\mathcal{M}|$ | #Nodes | Seconds | #Nodes | Seconds |
| 20 | 4,701 | 214,119 | 0.35 | 13,869 | 0.34 |
| 19 | 6,463 | 599,798 | 2.54 | 20,529 | 0.35 |
| 18 | 9,038 | 1,776,157 | 8.18 | 31,442 | 0.45 |
| 17 | 12,879 | 9,040,719 | 51.46 | 49,880 | 0.59 |
| 16 | 18,978 | 2,968,677,772 | 21,031.44 | 82,607 | 0.92 |

### Table 5 Top enriched signatures in maximal patterns; $S_{min} = 20$

| KEGG Pathways | N | KEGG Pathways | N |
|----------------|---|---------------|---|
| RIBOSOME       | 884 | HUNTINGTONS_DISEASE | 274 |
| PARKINSONS_DISEASE | 261 | OXIDATIVE_PHOSPHORYLATION | 257 |
| ALZHEIMERS_DISEASE | 251 | CARDIAC_MUSCLE_CONTRACTION | 158 |
| VIRAL_MYOCARDITIS | 21 | TIGHT_JUNCTION | 18 |
| AUTOIMMUNE_THYROID_DISEASE | 17 | SPLICEOSOME | 16 |

| GO Biological Process | N | GO Molecular Function | N |
|-----------------------|---|-----------------------|---|
| VIRAL_GENE_EXPRESSION | 888 | RRNA_BINDING | 511 |
| RNA_CATABOLIC_PROCESS | 886 | ELECTRON_TRANSFER_ACTIVITY | 223 |
| TRANSLATIONAL_INITIATION | 884 | MRNA_BINDING | 205 |
| PROTEIN_TARGETING | 884 | NADH_DEHYDROGENASE_ACTIVITY | 133 |
| CYTOPLASMIC_TRANSLATION | 808 | ANTIGEN_BINDING | 113 |
| RIBOSOME_BIOGENESIS | 777 | 5S_RRNA_BINDING | 69 |
| RIBOSOME_ASSEMBLY | 722 | CADHERIN_BINDING | 39 |