Efficiently mining rich subgraphs from vertex-attributed graphs

Riyad Hakim
Department of Computer Science
North Dakota State University
Fargo, North Dakota, USA
riyad.hakim@ndsu.edu

Saeed Salem
Department of Computer Science
North Dakota State University
Fargo, North Dakota, USA
saeed.salem@ndsu.edu

ABSTRACT

With the rapid collection of large network data such as biological networks and social networks, it has become very important to develop efficient techniques for network analysis. In many domains, additional attribute data can be associated with entities and relationships in the network, where the network data represents relationships among entities in the network and the attribute data represents various characteristics of the corresponding entities and relationships in the network. Simultaneous analysis of both network and attribute data results in detection of subnetworks that are contextually meaningful. We propose an efficient algorithm for enumerating all connected vertex sets in an undirected graph. Extending this enumeration approach, an algorithm for enumerating all maximal cohesive connected vertex sets in a vertex-attributed graph is proposed. To prune search branches that will not yield maximal patterns, we also present three pruning techniques for efficient enumeration of the maximal cohesive connected vertex sets. Our comparative runtime analyses show the efficiency and effectiveness of our proposed approaches. Gene set enrichment analysis shows that protein-protein interaction subnetworks with similar cancer dysregulation attributes are biologically significant.

Availability: The implementation of the algorithm is available at http://www.cs.ndsu.nodak.edu/~ssalem/richsubgraphs.html

KEYWORDS

Cohesive connected subgraphs, attributed graphs, enumeration algorithms

ACM Reference Format:
Riyad Hakim and Saeed Salem. 2020. Efficiently mining rich subgraphs from vertex-attributed graphs. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (BCB '20), September 21–24, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3388440.3412423

1 INTRODUCTION

Recent studies have focused on learning from graphs. Extracting and analyzing highly interacting vertices in dense subgraphs and percolated cliques improve understanding of functional building blocks in complex systems. Vertices and edges in graphs can be annotated with additional information, giving rise to the notion of rich graphs. The integration of attribute similarity while mining connected subgraphs with certain topological properties allows for the identification of context-specific connected subgraphs. To discover interesting patterns in a graph, it is often necessary to find connected subgraphs that satisfy some additional constraints. In many applications, the constraint may be defined based on the number of common attributes shared by the vertices of a subgraph. For example, in a protein–protein interaction (PPI) network, a connected set of proteins that are dysregulated in certain types of diseases might be particularly interesting to medical scientists [10]. Mining biologically relevant interaction subnetworks by integrating gene expression similarity with protein-protein interaction networks has been shown to improve identification of biological functional modules [6], subnetwork biomarkers [3, 5], and active subnetworks [4, 7]. In a social network, a connected group of people who share certain common interests might be the target of specific marketing campaigns [11]. In all of these applications, the goal is to mine sets of vertices such that vertices in each set are connected and have high attribute similarity; such vertex sets are referred to as cohesive connected vertex sets. A challenging aspect of mining these cohesive patterns is that the number of connected vertex sets may increase exponentially with the number of vertices in the graph and thus the enumeration process must be efficient in both time and memory usage.

In pattern mining, anti-monotone constraints are useful for pruning out the search space because of the fact that if a pattern does not satisfy an anti-monotone constraint, then none of its super-patterns will satisfy that constraint. We say that a vertex set and its induced subgraph are rich with respect to a given anti-monotone constraint, if and only if the vertex set satisfies that anti-monotone constraint. All the subgraphs of a rich subgraph are also rich due to the down-closure property of the constraint, resulting in an exponential number of rich subgraphs that have high overlap. It is often desirable to mine a representative set of all rich subgraphs to facilitate downstream analysis of these patterns. In a given graph, a rich connected vertex set and its induced subgraph are *maximal* if the vertex set is not a subset of another rich connected vertex set. The set of all maximal rich connected vertex sets is much smaller than the set of all rich connected vertex sets. Moreover, enumerating just maximal rich connected vertex sets, instead of all rich connected vertex sets, may enable us to prune large branches of the search space by developing pruning strategies with respect to the given anti-monotone constraint.

To enumerate all maximal rich connected vertex sets in a given graph, Maxwell et al. [10] introduced the BDDE algorithm. The BDDE algorithm relies on the given anti-monotone constraint to

BCB '20, September 21–24, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7964-9/20/09…$15.00
https://doi.org/10.1145/3388440.3412423

prune out the search space by considering all subsets of a connected vertex set for enumeration before the vertex set itself is considered. However, the limitation of BDDE is that it requires exponential space in the number of vertices in the graph and is not suitable for enumerating all maximal rich connected vertex sets in a large graph [10]. Wernicke [16] introduced the ESU algorithm for enumerating connected vertex sets. Starting from a single vertex, ESU exhaustively search the graph in depth-first order for connected vertex sets. Even though ESU is an efficient algorithm for connected vertex set enumeration, the algorithm has not been employed for mining rich connected vertex sets from a vertex-attributed graph. Moreover, the algorithm does not provide any mechanism to efficiently check the maximality of leaf nodes in the search tree. An approach to mining all maximal rich connected vertex sets using ESU would be to maintain a list of encountered maximal rich connected vertex sets and every time a new leaf node is encountered, check if that leaf connected vertex set has a superset or subset in the maximal list and update the list accordingly. Given the large number of maximal rich connected vertex sets and the much larger number of leaf nodes, the ESU algorithm would be inefficient for the purpose of mining maximal rich connected vertex sets. Another algorithm for enumerating all connected vertex sets is the TGE algorithm introduced by Uno [15]. Starting by selecting a single vertex r, TGE enumerates all connected vertex sets that include r. To do this, the algorithm chooses a vertex v adjacent to r and partitions the search space of connected vertex sets that include r into those that include v by unifying the vertex v with r and those that do not include v by removing the vertex v. The algorithm recursively carries out the process until the vertex r does not have any adjacent vertex and then outputs the set of unified vertices at the leaf node as a connected vertex set. The TGE algorithm also has not been employed for mining attributed graphs. Moreover, it does not provide any mechanism to efficiently check the maximality of those leaf nodes in the search tree and would be inefficient for the purpose of mining maximal rich connected vertex sets. Recently, Alokshiya et al. [1] introduced the RS-SP algorithm for enumerating connected vertex sets. The RS-SP algorithm has been used to enumerate maximal rich connected vertex sets. However, since RS-SP is based on the reverse search principle introduced by Avis and Fukuda [2], the algorithm needs to check all neighbors of the current connected vertex set to find valid extensions of the current connected vertex set and thus checking for invalid extensions impacts the overall performance of the algorithm. The RS-SP algorithm has been shown to outperform the BDDE and the TGE algorithms on mining all connected vertex sets [1].

In this paper, we propose an algorithm to enumerate all connected vertex sets in an undirected graph that takes linear time per output and linear space in the number of vertices in the graph. We extend this enumeration approach and propose an algorithm to enumerate all maximal cohesive connected vertex sets in a vertex-attributed graph. We also present three pruning techniques for fast enumeration of the cohesive connected vertex sets. We demonstrate the efficiency and effectiveness of our proposed approach and the pruning techniques on a protein-protein interaction network with gene expression dysregulation in multiple cancer types as attributes.

2 METHOD

Let $G = (V, E)$ be an *undirected graph*, where $V = \{1, \ldots, n\}$ is the vertex set and E is the edge set of graph G. The number of vertices $|V|$ is called *order* of the graph G. For a vertex set $U \subseteq V$, its *induced subgraph* $G[U]$ is the graph whose vertex set is U and whose edge set consists of all of the edges in E that have both endpoints in U. We refer to an induced subgraph that is connected as a *connected subgraph*. A vertex set $U \subseteq V$ is a *connected vertex set* if its induced subgraph $G[U]$ is connected. The *adjacent vertex set* of a vertex $v \in V$, denoted $Adj(v)$, is the set of all vertices adjacent to vertex v. The *open neighborhood* of a vertex set $U \subseteq V$, denoted $N_{op}(U)$, is the set of all vertices from $V \setminus U$ that are adjacent to at least one vertex in U [16]. The $N_{op}(U)$ can be obtained by taking union of adjacent vertex sets of all vertices in U and then excluding the vertices that are in U from the union.

$$N_{op}(U) = \{\bigcup_{v \in U} Adj(v)\} \setminus U$$

2.1 Mining Connected Vertex Sets

Problem Definition 1. *Given an undirected graph $G = (V, E)$, enumerate all connected vertex sets in G. The set of all connected vertex sets in G is defined as:*

$$\mathcal{CVS}(G) = \{U : U \subseteq V,\ U \text{ is connected}\}$$

The set of all connected vertex sets forms a search graph whose search nodes represent connected vertex sets and there is an edge between two search nodes if the two search nodes differ by only one vertex. An efficient approach for enumerating all connected vertex sets is to only traverse a minimum spanning tree of this search graph. A key feature of such an algorithm is to avoid visiting the same search node multiple times and to devise a strategy for obtaining a vertex set by extending only one vertex set. This is achieved by carefully defining the search nodes that can be reached from a given search node. This is done by designating a set of neighbors to extend the current vertex set.

Algorithm 1 shows the pseudo-code of the Miner algorithm that enumerates all connected vertex sets in an undirected graph. All list-like data structures are assumed to be indexed from 1. The Miner algorithm maintains two globally accessible lists U and N. The list U represents the current connected vertex set being enumerated, and N is the neighbor list of U. The open neighborhood of the connected vertex set U is essentially the neighbor list N minus the current connected vertex set U, i.e., $N_{op}(U) = N \setminus U$. Both U and N can be implemented with a fixed-size array and a *size* variable indicating the actual number of elements in the array. The list U can contain at most n elements, and N can contain at most $n - 1$ elements, where n is the number of vertices in the graph.

We start exploring the given graph from each vertex in the graph (lines 3-7). We denote the first vertex in the enumeration subtree as an *anchor vertex*. Starting from an anchor vertex u, we explore vertices in the graph in depth-first order, and when we explore a new vertex, we output the path from the anchor vertex to the newly explored vertex as a connected vertex set. This way, we enumerate all connected vertex sets that include vertex u. Next we move to another anchor vertex v to start a new search branch. Starting from

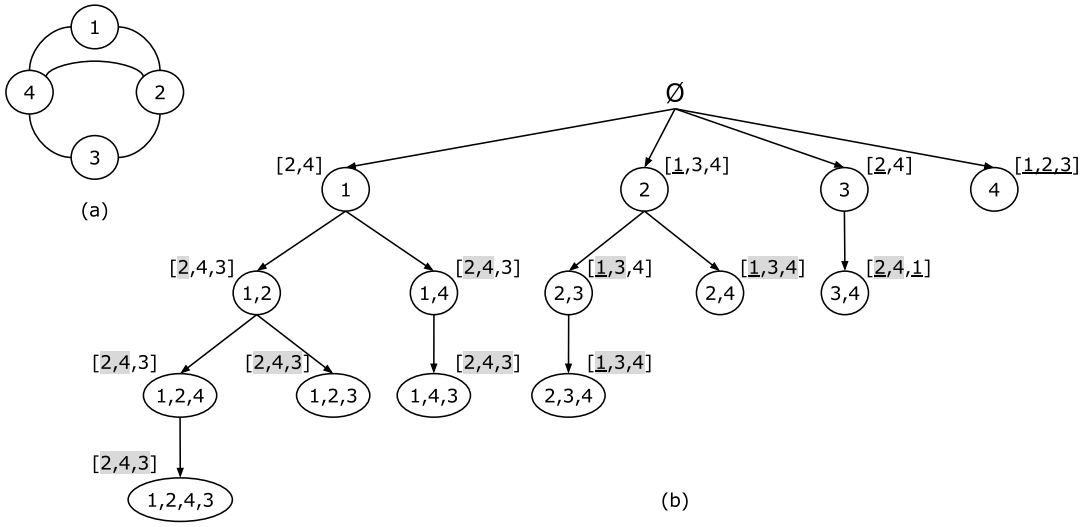

Figure 1: Connected vertex set enumeration. (a) a sample graph, (b) enumeration tree of the sample graph. Every node represents a unique connected vertex set.

anchor vertex v, we again explore vertices in the graph in depth-first order. However, this time we only enumerate connected vertex sets that include vertex v but not vertex u. We skip connected vertex sets with vertex u as those are already enumerated under the search branch rooted at anchor vertex u. The second exploration process is essentially the same as the first one except that this time when we encounter vertex u, we do not recursively explore the search subtree. In general, every time a vertex v is selected for anchoring, all vertices smaller than v are considered previously anchored (*pre-anchored*), and the depth-first exploration will enumerate all connected vertex sets that include the anchor vertex and none of the pre-anchored vertices.

For the depth-first exploration of the graph, we maintain just a single list N that stores both explored and unexplored neighbors. When we select a vertex from N to explore, instead of removing it from N, we just maintain an index variable *start* that marks the beginning of the unexplored neighbors in N, which is very efficient in comparison with maintaining two separate lists. Each vertex located before the *start* index in N is an *explored neighbor* and each vertex located at or after the *start* index in N is an *unexplored neighbor*. An unexplored neighbor v in N is considered *explored* if either U was extended by v or the extension was skipped because v is a pre-anchored neighbor (lines 12-15). Each unexplored neighbor in N that is larger than the anchor vertex is considered a *valid neighbor* for extension. Extending U only with valid neighbors eliminates the need to check for duplicate connected vertex sets.

Figure 1 shows the enumeration tree of our approach for a sample graph. We refer to the vertices of the enumeration tree as nodes, to avoid confusion with the vertices of the graph. Nodes are labeled with connected vertex set U. Beside each node, we also show the corresponding neighbor list N in which explored neighbors are in gray background, pre-anchored neighbors are underlined, and all other neighbors are valid neighbors.

Algorithm 1: Miner

Input : $G = (V, E)$: An undirected graph
Output: All connected vertex sets in G

```
1  U ← [ ], N ← [ ]
2  Visited[1] ← false, ..., Visited[n] ← false
3  for each vertex v ∈ V do
4      Visited[v] ← true
5      Append(U, v)
6      DepthFirstExplore(v, 1)
7      Visited[v] ← false
8  function DepthFirstExplore(lv, start)
9      output U
10     N_excl ← {u ∈ Adj(lv): Visited[u] = false}
11     AddNeighbors(N_excl)
12     for i ← start to Size(N) do
13         w ← N[i]
14         if w > U[1] then
15             Append(U, w)
16             DepthFirstExplore(w, i + 1)
17     Pop(U)
18     RemoveNeighbors(|N_excl|)
19 function AddNeighbors(Neighbors)
20     for each vertex u ∈ Neighbors do
21         Append(N, u)
22         Visited[u] ← true
23 function RemoveNeighbors(count)
24     for i ← 1 to count do
25         Visited[N[Size(N)]] ← false
26         Pop(N)
```

Complexity Analysis: An enumeration algorithm is said to have *linear delay* if the time between two consecutive outputs is bounded by a linear function of the input size, in the worst case. In our approach, the time between two consecutive outputs is dominated by the operations: adding neighbors (lines 10 – 11), removing neighbors (line 18), and the number of times the comparison $w > N[1]$ fails at a stretch (line 14) - each of which takes $O(n)$ time in the worst case scenario. Hence, the running time per output is $O(n)$. Therefore, the complexity of the algorithm is output polynomial time in the number of connected vertex sets. The algorithm uses three globally accessible lists/arrays U, N, and $Visited$ of a maximum size of n elements. Moreover, the space required for the local variables, e.g., lv, $start$, etc. is $O(n)$ as the depth of the enumeration tree is bounded by n. Hence, the total space required by the algorithm, excluding the space required for the input graph, is $O(n)$.

2.2 Mining Cohesive Connected Vertex Sets

Given an undirected graph $G = (V, E)$ and an attribute set $F = \{f_1, f_2, \cdots, f_d\}$, let $G = (V, E, f)$ be a *vertex-attributed graph* where $f : V \mapsto \mathbb{P}(F)$ is a function that maps each vertex to an element in the power set of attributes, indicating the associated attributes of the vertex. Each vertex $v \in V$ has an associated set of attributes $f(v)$. The association between the vertices and attributes can be represented by an $n \times |F|$ binary *attribute matrix*, $M = (m_{v,i})$ such that $m_{v,i} = 1$ if vertex v has the i^{th} attribute and $m_{v,i} = 0$ otherwise, for all $1 \leq v \leq n$ and $1 \leq i \leq |F|$.

A vertex set $U \subseteq V$ shares the i^{th} attribute f_i if all vertices in U have the i^{th} attribute in common, i.e., $f_i \in \cap_{v \in U} f(v)$. The *common attribute set* of a vertex set U, denoted $A(U)$, is the set of all common attributes the vertex set U shares, i.e., $A(U) = \cap_{v \in U} f(v)$. Given a user-defined threshold δ, we say that a vertex set U and its induced graph $G[U]$ are *cohesive*, if the number of the common attributes of the vertex set is at least δ, i.e., $|A(U)| \geq \delta$. The cohesive property is an anti-monotone constraint, which follows from the fact that if a vertex set is not cohesive then none of its supersets can be cohesive.

Given a vertex-attributed graph $G = (V, E, f)$ and a threshold δ for the minimum number of common attributes, the set of all cohesive connected vertex sets with respect to δ is defined as:

$$\mathcal{CCVS}(G, \delta) = \{U : U \subseteq V,\ U \textit{ is connected},\ |A(U)| \geq \delta\}$$

A major challenge for mining all cohesive connected vertex sets is that the number of all cohesive connected vertex sets can be very large, specially when δ is small. Moreover, there is an inherent overlap between the cohesive connected vertex sets since all the subsets of a cohesive vertex set are also cohesive. Therefore, we propose to enumerate all maximal cohesive connected vertex sets.

A cohesive connected vertex set is maximal if and only if it does not have a super vertex set that is also a cohesive connected vertex set, i.e., a cohesive connected vertex set $U \subseteq V$ is maximal if and only if $(\nexists U' \subseteq V)[\ U' \supsetneq U \textit{ and } U' \in \mathcal{CCVS}(G, \delta)\ ]$. To check for the supersets of the vertex set, we only need to check the immediate supersets that result by extending the set U with a neighboring vertex. So, a cohesive connected vertex set U is maximal if and only if $\nexists v \in N_{op}(U)$ such that $|A(U \cup \{v\})| \geq \delta$.

Problem Definition 2. *Given a vertex-attributed graph $G = (V, E, f)$ and a threshold δ for the minimum number of common attributes, enumerate all maximal cohesive connected vertex sets in G with respect to δ. The set of all maximal cohesive connected vertex sets in G with respect to δ is defined as:*

$$\mathcal{MCCVS}(G, \delta) = \{U : U \subseteq V,\ U \textit{ is connected},\ |A(U)| \geq \delta,\ \textit{and } U \textit{ is maximal}\}$$

To enumerate all cohesive connected vertex sets, we could just use the Miner algorithm and perform constraint checking while adding a vertex to the connected vertex set U. However, as we are only interested in maximal cohesive connected vertex sets, we also need an efficient way to check if a node in the enumeration tree is maximal or not. It is obvious that a non-leaf node in the enumeration tree can never be maximal, as every non-leaf node must have at least one valid neighbor that can extend it while maintaining the given constraint, otherwise it would become a leaf node. So only a leaf node in the enumeration tree has the potential to be maximal. However, even a leaf node in the enumeration tree is not maximal if it can be extended with an already explored or pre-anchored neighbor while maintaining the given constraint. Using this principle, we can efficiently check the maximality of any node in the enumeration tree. We represent the maximality principle below:

Maximality Principle: *A cohesive connected vertex set U is maximal with respect to a given threshold δ, if and only if there is no vertex v in the open neighborhood of U, irrespective of whether the vertex is an explored, pre-anchored or valid neighbor, that could be added to U such that $|A(U \cup \{v\})| \geq \delta$.*

We cannot directly incorporate maximality checking using the maximality principle in existing enumeration algorithms such as ESU. The first problem with the ESU is that ESU removes the vertices from its neighbor set when they are added to the connected vertex set. The second problem with the ESU is that ESU never adds the pre-anchored neighbors to its neighbor set. But to determine the maximality of a given cohesive connected vertex set using maximality principle, all neighbors, including explored and pre-anchored neighbors, need to be present.

In the Miner algorithm, however, we do not remove the explored neighbors from its neighbor list N. Moreover, when we add neighbors to our neighbor list N, we include the pre-anchored neighbors. As a result, maximality principle can easily be incorporated into the Miner algorithm. So, we can address the problem of enumerating all maximal cohesive connected vertex sets by utilizing the Miner algorithm as a backbone of the enumeration process, imposing constraint checking while extending a connected vertex set, and reporting only those leaf nodes of the enumeration tree that cannot be extended with any neighbor in the open neighborhood of U maintaining the given constraint.

Figure 2 shows the enumeration tree of all maximal cohesive connected vertex sets for a sample graph with $\delta = 2$. Nodes are labeled with connected vertex set U. Beside each node, we also show the corresponding neighbor list N and below each search node we show the binary vector indicating the common attributes of the vertices of the search node.

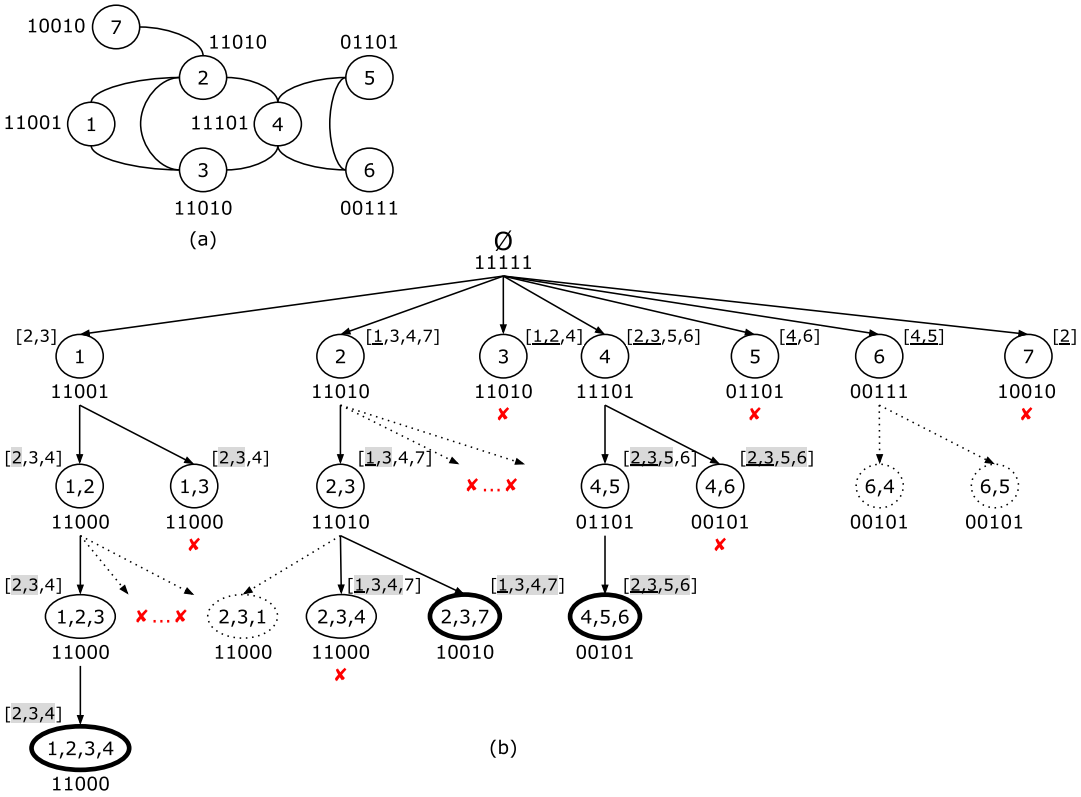

Figure 2: Maximal cohesive connected vertex set enumeration. (a) sample vertex-attributed graph, (b) enumeration tree of the sample graph with $\delta = 2$. The dotted portions are not part of the enumeration tree, they are for illustration purpose only. Every node with thick border represents a maximal cohesive connected vertex set. Nodes marked with X are pruned using pruning covered and pruning at level one techniques, and nodes marked with X...X are pruned using pruning rest technique.

Optimizing Neighbor Generation: We can further optimize the algorithm if, while adding the neighbors to N, we just add those unvisited neighbors to N that can extend U maintaining the given constraint. We can do this because if adding a neighbor to the current state of U does not generate a cohesive connected vertex set then adding that neighbor will not generate a cohesive connected vertex set after adding even more vertices to U.

For a large vertex-attributed graph and a relaxed cohesive constraint, exploring the entire enumeration tree of cohesive patterns is computationally expensive. Not all subtrees in the enumeration tree would generate maximal cohesive patterns. These futile subtrees constitute a large part of the enumeration tree, and thus early pruning of these subtrees results in a drastic reduction of the number of cohesive search nodes explored and improves the performance of the algorithm. We propose three pruning techniques that result in significant performance improvement while maintaining the completeness of the result.

Pruning Covered: Given a cohesive connected vertex set U, a neighbor $x \in N_{op}(U)$, and a threshold δ, we say that $U \cup \{x\}$ is a cohesive child of U if $|A(U \cup \{x\})| \geq \delta$. Let $U \cup \{x\}$ and $U \cup \{y\}$ be two cohesive children of U. We say that the cohesive child $U \cup \{x\}$ *precedes* another cohesive child $U \cup \{y\}$ if x is located before y in the neighbor list N. We also say that the cohesive child $U \cup \{x\}$ *covers* another cohesive child $U \cup \{y\}$ if $A(U \cup \{y\}) \subseteq A(U \cup \{x\})$.

Lemma 1. *If $U \cup \{x\}$ and $U \cup \{y\}$ are two cohesive children of connected vertex set U, $U \cup \{x\}$ precedes $U \cup \{y\}$, and $U \cup \{x\}$ covers $U \cup \{y\}$, then neither $U \cup \{y\}$ nor any descendant of $U \cup \{y\}$ is a maximal cohesive connected vertex set, and hence the search branch rooted at $U \cup \{y\}$ can be safely pruned.*

Proof. We begin with the first part. It is clear that $U \cup \{y\}$ is not a maximal cohesive connected vertex set as it has a cohesive connected superset $U \cup \{y\} \cup \{x\}$ because $A(U \cup \{y\} \cup \{x\}) = A(U \cup \{y\})$ and $U \cup \{y\}$ is cohesive. For the second part, we utilize the fact that once U is extended with y, $U \cup \{y\}$ is only extended with neighbors in N that succeeds y. As x precedes y and a vertex does not appear more than once in N, x will never be found in N after y. So no descendant of $U \cup \{y\}$ will include x in its vertex set. However, x could be added to every descendant of $U \cup \{y\}$ maintaining cohesiveness. So, none of the descendants of $U \cup \{y\}$ is a maximal cohesive connected vertex set. □

In Figure 2, the cohesive child $\{1, 3\}$ is pruned because it is covered by the preceding cohesive child $\{1, 2\}$ ($A(\{1, 3\}) \subseteq A(\{1, 2\})$).

Similarly, the search subtree rooted at $\{4, 6\}$ is pruned because it is covered by the preceding cohesive child $\{4, 5\}$.

Pruning at Level One: With a little change, we can apply the concept of pruning covered technique to level one of the enumeration tree where each node is a cohesive vertex set consisting of just a single vertex. Applying a similar approach used in the proof of lemma 1, it can be shown that given two adjacent vertices x and y, if both $\{x\}$ and $\{y\}$ are cohesive, $x < y$, and $\{x\}$ covers $\{y\}$ ($A(y) \subseteq A(x)$), then neither $\{y\}$ nor any descendant of $\{y\}$ is a maximal cohesive connected vertex set and the entire branch rooted at $\{y\}$ can be safely pruned. In Figure 2, node $\{3\}$ is pruned because it is covered by the smaller neighbor $\{2\}$. Similarly, nodes $\{5\}$ and $\{7\}$ are pruned because they are covered by nodes $\{4\}$ and $\{2\}$ respectively.

Pruning Rest: This technique is an extension of the pruning covered technique. Given a cohesive connected vertex set U, if U has a cohesive child $U \cup \{x\}$ that has the same set of attributes as its parent U, i.e., $A(U) = A(U \cup \{x\})$, then all cohesive children of U succeeding $U \cup \{x\}$ can be pruned as they all will be covered by the preceding cohesive child $U \cup \{x\}$. By applying this technique, we can prune the remaining cohesive children of U at once instead of applying the pruning covered technique to each of them separately. In Figure 2, all cohesive children of $\{1, 2\}$ succeeding $\{1, 2, 3\}$ are pruned because $\{1, 2, 3\}$ has the same set of attributes as its parent $\{1, 2\}$. Similarly, all cohesive children of $\{2\}$ succeeding $\{2, 3\}$ are pruned because $\{2, 3\}$ has the same set of attributes as its parent $\{2\}$.

Algorithm 2 shows the pseudo-code of the CSMiner algorithm that enumerates all maximal cohesive connected vertex sets. We start a search subtree for each cohesive vertex that cannot be pruned. Lines 5-7 handle level one pruning, and lines 21-23 handle the pruning covered strategy. Lines 26-27 prune rest of the succeeding cohesive children. Line 19 marks U as not maximal if the neighbor that is currently being explored could be added to U while maintaining cohesiveness. Similarly, line 31 marks U as not maximal if there is a previously explored neighbor in the open neighborhood of U that could be added to U without violating the cohesive constraint. In line 21 and line 29, we need to make sure that the vertex selected from N is not already an element of U, which can be checked in constant time as, except the anchor vertex, all vertices in U are added sequentially from N. Finally, the algorithm reports only those leaf nodes that do not have any cohesive super vertex set.

Algorithm 2: Cohesive Subgraph Miner (CSMiner)

Input : $G = (V, E, f)$: An undirected vertex-attributed graph,
δ: Minimum number of common attributes
Output: All maximal cohesive connected vertex sets in G

```
1  U ← [ ], N ← [ ]
2  Visited[1] ← false, ... , Visited[n] ← false
3  for each vertex v ∈ V do
4      if |A({v})| ≥ δ then
5          for each vertex u ∈ Adj(v) do
6              if u < v and A({v}) ⊆ A({u}) then
7                  goto line 3
8          Visited[v] ← true
9          Append(U, v)
10         DepthFirstExplore(v, 1)
11         Visited[v] ← false
12 function DepthFirstExplore(lv, start)
13     maximal ← true
14     N_excl ← {u ∈ Adj(lv): Visited[u] = false,
         |A(U ∪ {u})| ≥ δ}
15     AddNeighbors(N_excl)
16     for i ← start to Size(N) do
17         w ← N[i]
18         if |A(U ∪ {w})| ≥ δ then
19             maximal ← false
20             if w > U[1] then
21                 for each vertex x ∈ N[1 : (i − 1)] \ U do
22                     if A(U ∪ {w}) ⊆ A(U ∪ {x}) then
23                         goto line 16
24                 Append(U, w)
25                 DepthFirstExplore(w, i + 1)
26             if A(U) = A(U ∪ {w}) then
27                 break
28     if maximal then
29         for each vertex z ∈ N[1 : (start − 1)] \ U do
30             if |A(U ∪ {z})| ≥ δ then
31                 maximal ← false
32                 break
33         if maximal then
34             output U
35     Pop(U)
36     RemoveNeighbors(|N_excl|)
```

3 RESULTS

We evaluated the performance of our algorithm with respect to existing algorithms for enumerating connected vertex sets on both random and real graphs. We also evaluated the performance of our algorithm for enumerating maximal cohesive connected vertex sets on a large real vertex-attributed graph. All experiments were conducted on a machine with Intel Xeon E5-2670 v2 processor and 32 GB memory, running linux operating system. We implemented our algorithms in C++. To evaluate the runtime of existing algorithms, we used the C/C++ implementations provided by the respective authors.

3.1 Mining Connected Vertex Sets from Random Graphs

To compare the runtime of our Miner algorithm on randomly generated graphs, we chose two of the fastest existing algorithms (as shown in [1]) for enumerating connected vertex sets, i.e., RS-SP and TGE. We ran the algorithms on randomly generated graphs of different orders and densities. We generated random graphs according to the $G(n, m)$ Erdos-Renyi model. Results in Figure 3 show that our algorithm runs significantly faster than both RS-SP and TGE. On average, the Miner algorithm was three times faster than

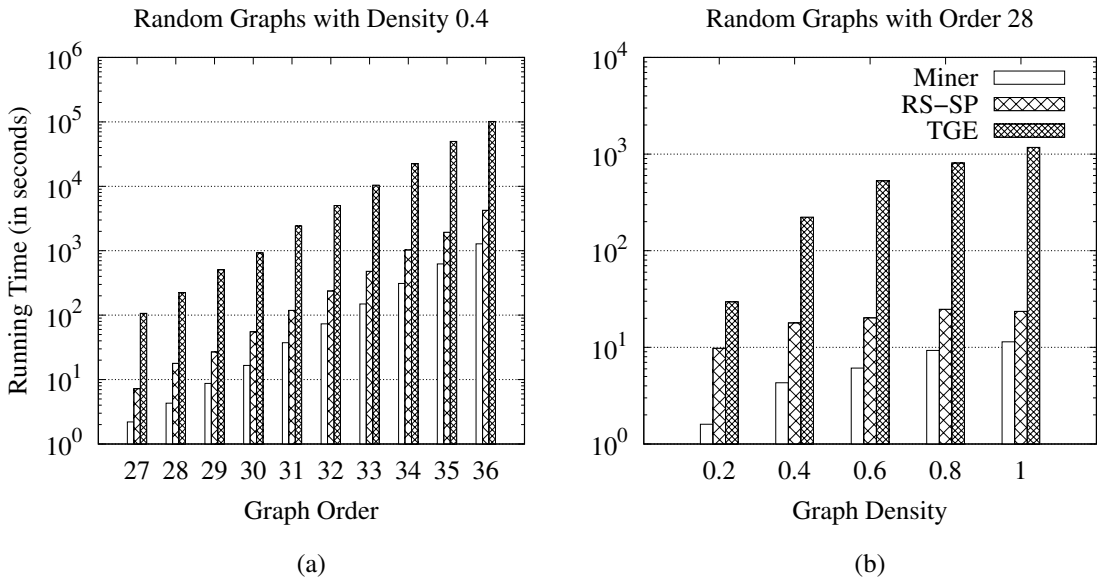

Figure 3: Runtime for enumerating connected vertex sets in random graphs. (a) Graphs of varying orders with fixed density, (b) Graphs of varying densities with fixed order.

the RS-SP algorithm across all input random graphs. The TGE algorithm was too slow in comparison with the proposed algorithm.

3.2 Mining Connected Vertex Sets from Real Graphs

We compared the runtime of our Miner algorithm on real graphs with that of RS-SP. We applied the algorithms on real chemical graphs from the network data repository [13]. Table 1 shows again that our algorithm runs much faster than RS-SP. Since the maximum number of chemical bonds an atom can form is restricted by the number of valence electrons in the atom's outer shell, large chemical graphs are usually not dense. We chose 10 enzyme graphs with orders between 40 and 49 and densities between 0.079 and 0.101. On average, the Miner algorithm was 5.75 times faster than RS-SP across all input real graphs.

3.3 Mining Cohesive Connected Vertex Sets from Real Graph

To evaluate the performance of our CSMiner algorithm, we used the BioGRID human protein-protein interaction (PPI) network, version 3.5.182 [12]. The network has 24, 052 genes (vertices) and 389, 245 physical interactions (edges). We used the gene dysregulation binary profile across 13 cancer types as attributes of the vertices. We generated the profiles from the gene expression data in 13 cancer types [8]. Genes that are not present in the cancer data were given profiles of all zeros. We applied the algorithm on the vertex-attributed graph with δ values from 1 to 13. For δ values 1, 2, 3, and 4, the algorithm took only 10.63, 4.95, 2.54, and 1.31 seconds respectively, and for $\delta \geq 5$, the algorithm took less than 1 second to complete the enumeration.

Table 1: Runtime for enumerating connected vertex sets in real chemical graphs

Graph ID	$\|V\|$	Density	$\|\mathcal{CVS}(G)\|$ (in billions)	Miner (s)	RS-SP (s)
E-g564	40	0.095	5.06	50.74	255.20
E-g303	41	0.101	22.53	238.53	1,260.39
E-g308	42	0.095	31.47	310.62	1,769.07
E-g538	43	0.098	128.26	1,320.20	7,587.90
E-g569	44	0.090	135.61	1,500.56	8,546.11
E-g101	45	0.089	161.88	1,599.11	9,874.30
E-g117	46	0.087	321.14	3,276.33	19,813.74
E-g195	47	0.085	372.32	3,620.53	20,289.20
E-g171	48	0.088	1,276.57	14,497.35	79,858.06
E-g300	49	0.079	1,987.98	19,096.20	127,369.58

3.3.1 Effectiveness of Pruning Techniques. To evaluate the effectiveness of the proposed pruning techniques, we also ran the CSMiner algorithm with each pruning technique disabled one at a time. Figure 4 shows the effects of disabling the pruning techniques for δ values from 1 to 4. With the pruning covered technique disabled, the algorithm did not finish in 48 hours, hence omitted in the figure. For a very relaxed constraint, the number of cohesive connected vertex sets in a graph may be exponentially large in the number of vertices in the graph. To mine all maximal cohesive connected vertex sets without pruning, the algorithm will explore the entire search tree of cohesive connected vertex sets. However, with pruning, large branches of this search tree are pruned based on the pruning criteria, hence the improved performance. It is evident from our experiment that the proposed algorithm with constraint-specific pruning strategies improves the performance.

Table 2: Enrichment analysis of extracted gene sets

δ	Pattern#	Avg. Size	Avg. Density	Hallmark%	CM%	BP%	CC%	MF%
2	196	179.07	0.336	67.86	70.92	91.84	83.16	97.96
3	578	72.88	0.266	70.24	75.95	93.60	83.56	93.60
4	865	48.11	0.226	76.18	81.73	92.02	87.51	93.06
5	801	39.52	0.197	83.77	87.39	94.38	91.89	94.76
6	519	32.82	0.192	88.44	91.33	95.76	93.45	94.99
7	242	26.79	0.209	88.84	94.63	95.87	94.21	94.21
8	95	21.41	0.242	87.37	92.63	95.79	95.79	95.79
9	38	16.08	0.296	89.47	92.11	100.00	94.74	100.00
10	15	11.80	0.342	100.00	93.33	100.00	93.33	100.00

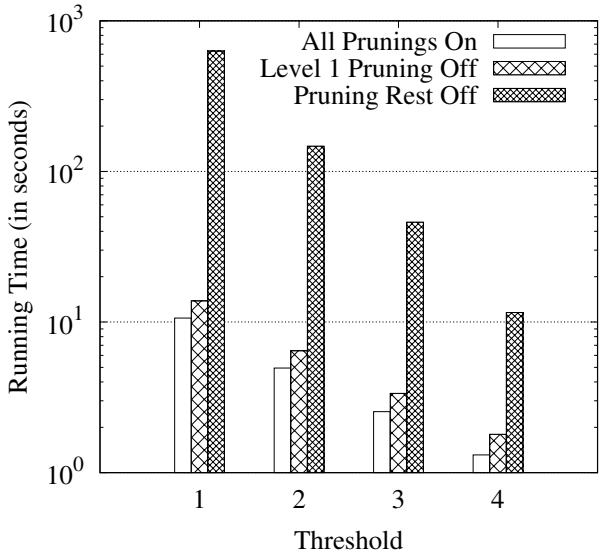

Figure 4: Effectiveness of Pruning Techniques.

3.3.2 Analysis of Extracted Gene Sets. We analyzed the topological properties and biological significance of the gene sets (maximal cohesive connected vertex sets) extracted from the BioGRID network by the CSMiner algorithm for δ values from 2 to 10. The goal of the enrichment analysis is to check if the extracted gene sets have association with known biological processes, molecular functions, or disease phenotypes. The enrichment was assessed by the overlap of the extracted gene sets with predefined collection of gene sets. For this analysis, we only assessed the cohesive patterns with at least three genes. As predefined collection of gene sets, we used the Hallmark, Cancer Modules (CM), GO Biological Process (BP), GO Cellular Component (CC), and GO Molecular Function (MF) gene set collections from Molecular Signatures Database (MSigDB) [9, 14]. These collections have been grouped together based on their involvement in the same biological pathway, molecular process, molecular function, or by proximal location on a chromosome.

An extracted gene set is considered enriched with a signature collection if it has a significant overlap with at least one gene set in that collection. The overlap is assessed with an over-representation test. The hypergeometric test with p-value = 0.05 is used for over-representation testing. Table 2 shows the total number, average size, and average density of the extracted gene sets along with the percentage of the patterns that are enriched with the signature gene set collections. The results show that the average size of the extracted gene sets decreases when δ is increased. After an initial decrease for δ values up to 6, the average density of the extracted gene sets increases when δ is increased. This is expected as the subnetworks whose genes are dysregulated in a large number of cancers tend to be smaller and denser compared with the subnetworks dysregulated in a small number of cancers.

4 DISCUSSION AND CONCLUSIONS

We proposed an algorithm for enumerating all connected vertex sets in a graph. The algorithm constructs an enumeration tree where each connected vertex set appears only once. Extending this enumeration approach, we proposed an algorithm to enumerate all maximal cohesive connected vertex sets in a given vertex-attributed graph. We also incorporated pruning strategies into the algorithm that drastically reduced the number of explored search nodes. Experiments conducted on both real and synthetic data sets show the effectiveness of the proposed approaches. Gene set enrichment analysis shows that the gene sets extracted are biologically significant. In the future, we would like to develop a parallel implementation of the algorithm.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. RII Track-2 FEC 1826834.

REFERENCES

[1] Mohammed Alokshiya, Saeed Salem, and Fidaa Abed. 2019. A linear delay algorithm for enumerating all connected induced subgraphs. *BMC Bioinformatics* 20, 12 (2019), 319.
[2] David Avis and Komei Fukuda. 1996. Reverse search for enumeration. *Discrete Applied Mathematics* 65, 1 (1996), 21–46.
[3] Salim A Chowdhury, Rod K Nibbe, Mark R Chance, and Mehmet Koyutürk. 2011. Subnetwork state functions define dysregulated subnetworks in cancer. *Journal of Computational Biology* 18, 3 (2011), 263–281.
[4] Han-Yu Chuang, Eunjung Lee, Yu-Tsueng Liu, Doheon Lee, and Trey Ideker. 2007. Network-based classification of breast cancer metastasis. *Mol Syst Biol.* 3 (2007), 140.

[5] Han-Yu Chuang, Eunjung Lee, Yu-Tsueng Liu, Doheon Lee, and Trey Ideker. 2007. Network-based classification of breast cancer metastasis. *Molecular systems biology* 3, 1 (2007).

[6] Elisabeth Georgii, Sabine Dietmann, Takeaki Uno, Philipp Pagel, and Koji Tsuda. 2009. Enumeration of condition-dependent dense modules in protein interaction networks. *Bioinformatics* 25, 7 (2009), 933–940.

[7] Trey Ideker, Owen Ozier, Benno Schwikowski, and Andrew F. Siegel. 2002. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics* 18, Suppl 1 (2002), S233–40.

[8] Wei Jiang, Ramkrishna Mitra, Chen-Ching Lin, Quan Wang, Feixiong Cheng, and Zhongming Zhao. 2016. Systematic dissection of dysregulated transcription factor–miRNA feed-forward loops across tumor types. *Brief Bioinform.* 17, 6 (2016), 996–1008.

[9] Arthur Liberzon, Chet Birger, Helga Thorvaldsdóttir, Mahmoud Ghandi, Jill P. Mesirov, and Pablo Tamayo. 2015. The Molecular Signatures Database Hallmark Gene Set Collection. *Cell Systems* 1, 6 (2015), 417 – 425.

[10] Sean Maxwell, Mark R Chance, and Mehmet Koyutürk. 2014. Efficiently Enumerating All Connected Induced Subgraphs of a Large Molecular Network. In *International Conference on Algorithms for Computational Biology (Lecture Notes in Computer Science, vol 8542)*. Springer, Cham, 171–182.

[11] Flavia Moser, Recep Colak, Arash Rafiey, and Martin Ester. 2009. Mining Cohesive Patterns from Graphs with Feature Vectors.. In *SDM*, Vol. 9. 593–604.

[12] Rose Oughtred et al. 2018. The BioGRID interaction database: 2019 update. *Nucleic Acids Research* 47, D1 (11 2018), D529–D541.

[13] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. http://networkrepository.com

[14] Aravind Subramanian, Pablo Tamayo, Vamsi K. Mootha, Sayan Mukherjee, Benjamin L. Ebert, Michael A. Gillette, Amanda Paulovich, Scott L. Pomeroy, Todd R. Golub, Eric S. Lander, and Jill P. Mesirov. 2005. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences* 102, 43 (2005), 15545 – 15550. http://www.pnas.org/content/102/43/15545

[15] Takeaki Uno. 2015. Constant Time Enumeration by Amortization. In *Algorithms and Data Structures. WADS 2015. Lecture Notes in Computer Science, Vol 9214*, Frank Dehne, Jörg-Rüdiger, and SackUlrike Stege (Eds.). Springer, Cham, 593–605.

[16] S. Wernicke. 2006. Efficient Detection of Network Motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 3, 4 (2006), 347–359.