Mining cohesive patterns from graphs with feature vectors

Flavia Moser, Recep Colak, Arash Rafiey, Martin Ester
fmoser, rcolak, arafieyh, ester@cs.sfu.ca
Simon Fraser University, Canada
October 10, 2008

Abstract
The increasing availability of network data is creating a great potential for knowledge discovery from graph data. In many applications, feature vectors are given in addition to graph data, where nodes represent entities, edges relationships between entities, and feature vectors associated with the nodes represent properties of entities. Often features and edges contain complementary information, and in such scenarios the simultaneous use of both data types promises more meaningful and accurate results. Along these lines, we introduce the novel problem of mining cohesive patterns from graphs with feature vectors, which combines the concepts of dense subgraphs and subspace clusters into a very expressive problem definition. A cohesive pattern is a dense and connected subgraph that has homogeneous values in a large enough feature subspace. We argue that this problem definition is natural in the identifying small communities in social networks and functional modules in Protein-Protein interaction networks. We present the algorithm CoPaM, which exploits various pruning strategies to efficiently find all maximal cohesive patterns. Our theoretical analysis proves the correctness of CoPaM, and our experimental evaluation demonstrates its effectiveness and efficiency.

1 Introduction
Graphs provide a natural representation of important real life networks such as social networks and biological networks. Recently, such network data has become increasingly available. While earlier analysis methods focused on graph properties such as degree distribution, diameter and simple graph patterns such as cliques, more recent analysis methods aim at finding more sophisticated patterns and structures in graphs. In social network analysis, e.g., online social network data is being analyzed to detect communities that can be used for more targeted delivery of online advertisements. In systems biology researchers want to find functional modules in protein interaction networks which can serve as the basis of computer-aided drug design.

Most of the existing methods such as graph partitioning [17] and quasi-clique finding [16] work on graph data only. However, in many applications more informative graphs are given, where nodes represent entities, edges relationships between entities, and feature vectors associated with the nodes represent entity properties such as demographic attributes of customers and expression data of genes. Often features and edges contain complementary information, i.e. neither the relationships can be derived from the attributes nor vice versa. In such scenarios the simultaneous use of both data types promises more meaningful and accurate results. Joint cluster analysis [4] aims at partitioning a single table with attributes and relationships into connected components with similar attribute values, a more specialized approach specifically targeting graphs with features. [18] introduces a spectral clustering method combining attribute and relationship data to partition the input. While these approaches can exploit attribute data and graph data, they cannot ensure that the discovered clusters are dense and connected, since they have to partition the entire graph. Furthermore, the identified clusters cannot overlap.

Integrating the concepts of dense subgraphs and subspace clusters, this paper introduces the novel problem of finding cohesive patterns. We define a cohesive pattern as a connected subgraph whose density exceeds a given threshold and which has, in a large enough subspace, homogeneous feature values. Different from graph partitioning methods and similar to frequent-pattern mining methods, cohesive patterns can overlap and do not have to cover the entire dataset. Furthermore, the number of patterns does not need to be specified in advance. A major criticism of pattern mining is the large number of patterns produced, in our case the number of dense connected subgraphs can be extremely high. Integrating constraints on the feature vectors reduces the number of patterns substantially and adds additional meaning to the identified patterns. Our algorithm effectively prunes the search space by simul-
taneously using the constraints on the feature vectors, density and connectedness and has therefore to consider only a small portion of the large number of subgraphs.

1.1 Motivational applications We use the following applications from social network analysis and systems biology to motivate our problem definition. In social network analysis, one of the most important tasks is the identification of communities [22], i.e. groups of people that have strong social interactions and share some interest. Communities like sports clubs or research groups have the following characteristic properties: The members of a community know each other, i.e. have many edges between them such that information can be exchanged and flow within the community. Members of a community are expected to have similar feature values in the subspace on which they are based, e.g. features related to the personal or professional life. Communities can overlap, since a person can be, e.g., member of a sports club and a research lab. The number of communities is not known in advance. Finally, not every person has to be part of a community. Figure 1 shows an example of a social network of computer science students where students are associated with two features, their favorite sports and their study focus. The first community consists of soccer players, united by their favorite sports, and the second one contains students who share logic as their study focus. Student A belongs to both communities. Students D, E, F are not part of any of the communities, because they do not share any interest (research or sports) with their friends (connected nodes).

In systems biology, one applications of our problem definition is the identification of functional modules, i.e. groups of genes that are involved in a specific cellular process. Initial attempts were restricted to the use of a single data type such as gene expression or protein interaction data. However, each of these data types describes only one specific aspect of the cellular system and fails to characterize the system as a whole. Most cellular functions are carried out by group of proteins, that highly interact with each other, but loosely interact with the rest of the proteins. Therefore, a functional module forms a dense and connected component in the interaction network, with complexes having the highest densities followed by pathways with a somewhat lower density. Functional modules are also characterized by similar expression patterns of their genes (that code the proteins of the module), though not in all conditions but only in a subset, because many proteins perform different functions in different tissues and during different developmental stages.

1.2 Main contributions Our main contributions are as follows:

- We introduce the novel problem of mining cohesive patterns, which integrates the concepts of dense subgraphs and of subspace clusters in a feature space.
- We developed the algorithm CoPaM (Cohesive Pattern Miner) that efficiently finds the set of all maximal cohesive patterns.
- We provide a theoretical analysis giving insights into what makes the problem so hard and prove the correctness of the CoPaM algorithm.
- We run experiments on social network and biological datasets demonstrating the meaningfulness of cohesive patterns and show the efficiency and scalability of CoPaM.

Overview: The rest of the paper is organized as follows. Section 2 reviews related work. In section 3, the problem of mining cohesive patterns is introduced. CoPaM - Cohesive Pattern Miner - is presented in section 4. A theoretical analysis of CoPaM can be found in Section 5. Section 6 reports the results of our experimental evaluation. Section 7 concludes the paper with a summary and interesting directions for future research.

2 Related Work
The topic of mining dense graphs has recently received a lot of attention in the data mining community. The existing methods assume as input a collection of graphs and produce the frequent subgraphs that satisfy some
coherency or density constraint. The input graphs are undirected vertex-labelled graphs, and one of the key issues in graph mining is how to efficiently perform the graph isomorphism testing which plays a crucial role in the counting of the support of candidate graph patterns. \cite{21} presents a method to find all frequent maximal cliques in a depth-first approach exploiting the anti-monotonicity properties of cliques and support. A more relaxed and more realistic density constraint requires only that the graph patterns are quasi-cliques, i.e. that every node has at least a specified percentage $\alpha$ of all possible edges within the pattern. \cite{16} and \cite{24} propose new search space pruning strategies for efficiently mining all frequent, and all closed frequent resp., quasi-cliques. In many applications, both clique and quasi clique constraints are very hard constraints and may cause missing subtle, but interesting patterns.

Different from the above methods, \cite{23} assumes a database of graphs with unique node identifiers, so that graph isomorphism is not an issue. They investigate the problem of mining all closed frequent graphs with edge connectivity at least $k$, where the edge connectivity is defined as the minimum cut size. The proposed CLOSECUT algorithm works well on datasets which contain mainly patterns with high support and low connectivity. The second approach, SPLAT, targets datasets containing mainly highly connected patterns. Relaxing the minimum support constraint, \cite{12} presents an algorithm to mine subgraphs that are dense, defined based on the size of the minimum cut, and exhibit correlated occurrence.

The above methods all work on large databases of labelled, relatively small graphs. Another line of research has investigated methods for finding dense subgraphs in a single large graph. However, finding dense subgraphs is a notoriously hard combinatorial problem, even to solve approximately (see, e.g., \cite{9}). In the absence of a minimum support constraint, such algorithms typically give up the goal of finding all dense subgraphs and resort to heuristics that efficiently find some of these subgraphs. For example, \cite{10} presents an algorithm based on a recursive application of shingling followed by a final clustering step. Graph partitioning algorithms such as normalized cut \cite{17} can be considered as another approach for efficiently finding some of the densest subgraphs. These algorithms partition the graph into components with small cut size, i.e. small weight of the edges between different components, which indirectly leads to dense components.

A variety of approaches has investigated the integrated mining of graphs with associated attributes. Multi-relational clustering algorithms, e.g. the PRM-based approach of \cite{19}, partition a database of multiple related tables, which can be understood as a graph with feature vectors, into a specified number of clusters optimizing an objective function such as the likelihood. Joint cluster analysis \cite{4} aims at partitioning a single table with attributes and relationships into connected components with similar attribute values in all dimensions, a more specialized approach specifically targeting graphs with features. While these approaches take into account attribute data and graph data, they do not ensure that the discovered clusters are dense and connected. The Co-Clustering method \cite{11} defines an integrated distance function incorporating both the similarity of attribute data and the network shortest path distance and then applies any distance-based clustering algorithm. Another integrated method that has been developed in the bioinformatics community is MATISSE \cite{20}, a probabilistic method that determines connected subnetworks in graphs (such as interaction networks) that exhibit high attribute (e.g., expression) similarity, without enforcing clusters to be dense and connected.

### 3 Problem definition

In this section we define the problem of mining cohesive patterns in feature graphs. First we give some definitions necessary to understand the problem definition.

**Definition 1. (Feature graph)** A feature graph is a graph $G = (V,E,D,F)$, in which $V = \{v_1, \ldots, v_n\}$ denotes the node set, $E \subseteq \{(v_i, v_j)|v_i, v_j \in V, v_i \neq v_j\}$ the edge set and $F : V \rightarrow D_1 \times \cdots \times D_k$ a feature function. $D = \{D_1, \ldots, D_k\}$ is called the feature space of $G$, $D' \subseteq D$ feature subspace.

Social network data or protein interaction networks in combination with gene expression data are examples for feature graphs, e.g. in Figure 1 the nodes in the network have a two-dimensional feature vector attached. This data can be formally represented as $G = (V,E,D,F)$, where $V = \{A, \ldots, I\}$, $E = \\{\{A,B\},\{A,C\},\ldots\}$, $D = \{\{soccerc, tennisc, squashc, dancingc\}\}$ and function $F$, e.g. $F(A) = (soccerc, logicc)$. Note that we denote with $G$ the overall graph, and with $G$ a subgraph in $G$. In this paper we are interested in finding cohesive patterns, i.e. subgraphs $G$ with certain properties. First, cohesive patterns have to be connected. Second, their density $d(G)$ needs to exceed a given threshold. In this paper, the density is defined as the cliquishness, which is the fraction of the number of edges divided by the number of possible edges. Third, we require the features of the nodes of $G$ to be cohesive in some feature subspace. In order to formalize this last constraint, we define a subspace cohesion function.
A **subspace cohesion function** \( s \) is a boolean function

\[
s : \mathcal{P}(V) \times \mathcal{P}(D) \times \mathbb{R} \rightarrow \{T, F\}
\]

which has as input a subset \( V \) of the node set \( V \), a subset \( D \) of the feature space \( D \), and a real number, such that

\[
(s(V, D, \theta_s) = T \land \exists D' \supseteq D : s(V, D', \theta_s) = T) \Rightarrow \\
(s(V, D', \theta_s) = T \Rightarrow D'' \subseteq D),
\]

i.e. the maximal cohesive feature subspace \( D \) for a subgraph with node set \( V \) is uniquely determined.

Furthermore, \( s \) is assumed to be anti-monotone, i.e.

\[
s(V, D, \theta_s) = T \Rightarrow s(V', D', \theta_s) = T \forall V' \subseteq V, D' \subseteq D
\]

\( \theta_s \) is called **subspace cohesion threshold**.

To illustrate the subspace cohesion function, consider again the example in Figure 1. All nodes in a community fulfill the cohesive pattern constraint (Definition 3). To prevent such a case and to enforce some stricter cohesion, we will constrain the size of \( D \) in the following definition, which reflect the properties of small communities in social networks and functional modules in protein interaction networks.

**Definition 2.** [**Cohesive pattern**] Given a feature graph \( G = (V,E,D,F) \) and the following parameters: subspace cohesion function \( s \), subspace cohesion threshold \( \theta_s \), dimensionality threshold \( \theta_{\text{dim}} \) and density threshold \( \alpha \). An induced subgraph \( G = (V,E,D), V \subseteq V, E = \{v_1, v_2 | v_1, v_2 \in V, \{v_1, v_2 \} \in E\}, D \subseteq D \), is called **cohesive pattern** if it satisfies the following three constraints:

1. **Subspace constraint:** \( G \) is homogeneous in \( D \subseteq D \), i.e. \( s(V, D, \theta_s) = true \) and \( |D| \geq \theta_{\text{dim}} \geq 1 \)

2. **Density constraint:** \( d(G) := \frac{2|E|}{V(|V|-1)} \geq \alpha \). (In this case \( G \) is also called \( \alpha \)-dense.)

3. **Connectivity constraint:** \( G \) is connected.

We call the density constraint, the subspace constraint, and the connectivity constraint together **cohesive pattern constraint (CP constraint)**. Furthermore, we call an edge cohesive if the induced subgraph of its corresponding nodes fulfills the CP constraint, otherwise non-cohesive.

**Definition 3.** [**Maximal cohesive pattern**] Given a feature graph \( G = (V,E,D,A) \) and a cohesive pattern \( G = (V,E,D) \). \( G \) is called **maximal cohesive pattern**, if \( \exists v \in V \) and \( \exists D' \subseteq D \) such that the graph \( G' = (V + v, E', D') \) induced by \( V \cup \{v\} \) is also a cohesive pattern. Furthermore, we require \( D \) to be maximal, i.e. \( \exists D' \subseteq D \), such that \( G = (V,E,D \cup \{d\}) \) is also a cohesive pattern.

This definition leads to the following problem definition in which we want to find maximal cohesive patterns.

**Cohesive pattern mining (CoPaM) problem**

Let \( G \) be a feature graph, \( s \) a subspace cohesion function, \( \theta_s \) a subspace cohesion threshold, \( \theta_{\text{dim}} \) a dimensionality threshold and \( \alpha \) a density threshold, the **Cohesive Pattern Mining (CoPaM) Problem** is to find the set of all maximal cohesive patterns of \( G \) wrt. the aforementioned parameters.

We briefly analyze the complexity of this problem definition. It is known that finding the maximum size clique of a graph is NP-complete [13]. In our problem definition, in the worst case, we need to find the maximum clique in the input graph. Hence it is NP-hard. However, we expect the size of the largest clique to be constant, therefore, in practice this part of the problem is feasible. In terms of the output, our problem can be in the worst case (all dense graphs are cohesive) reduced to the problem of counting all cliques from a graph which is known to be #P-Complete [14]. Again this theoretical worst case does typically not occur in real-world datasets and the runtimes reported in Section 6 show the practicality of our algorithm.

In the following, we define several properties of cohesive patterns and of nodes which are used in our algorithm.

**Definition 4.** [(Maximally) expanded-by-one] A cohesive pattern \( G = (\{v_1, \cdots, v_n\}, E, D) \) is called **expanded-by-one** if there exists at least one permutation \( \tau = (v_{i_1}, \cdots, v_{i_n}) \) over the nodes of \( G \) that induces a sequence \( (\{v_{i_1}, v_{i_2}\}, \ldots, G - \{v_{i_{n-1}}, v_{i_n}\}, G - v_{i_n}, G) \), such that all graphs in this sequence are cohesive patterns wrt. \( D \). If a cohesive pattern \( G \) cannot be extended by any neighboring node without violating the CP constraint, \( G \) is called maximally expanded-by-one.

The intuition behind this definition is the following. A graph \( G \) is expanded-by-one, if it can be generated by starting from two connected nodes and adding one connected node at a time, such that all resulting patterns are cohesive.

Not all cohesive patterns are expanded-by-one, as the example in Figure 2 shows. For \( \alpha = 0.41 \), the only node which can be removed without violation of the density constraint is \( c \). \( c \) is a bridge node which is defined in the following:
Figure 2: Example for cohesive pattern containing $\alpha$-critical node $c$ for $\alpha = 0.41$. $G_1 + c$ and $G_2 + c$ are merging candidates.

**Definition 5.** (Bridge node) $c$ is called **bridge node** if $G - c$ is disconnected.

Since $c$ is a bridge node, in $G - c$ the connectivity constraint is violated. However, $c$ is the only $\alpha$-removable node in this graph. In subsection 5, we show that this case can only occur for $\alpha < \frac{1}{2}$. Cohesive patterns which are not expanded-by-one, still have a very nice property. They can be decomposed into three subgraphs, namely two merging candidates and one expand-by-one part. The merging candidates are defined in the following.

**Definition 6.** [Merging candidate (node)] A cohesive pattern $G = (V, E, D)$ is called **merging candidate** if

$$\exists v \in V : \deg(v) < \deg(v') + 1 \forall v' \in V \setminus \{v\},$$

where $\deg(v)$ is the degree of $v$. $v$ is called **merging candidate node**.

For example, in Figure 2, two of the merging candidates are $G_1 + c$ and $G_2 + c$. We will need this concept of merging candidates in our algorithm which we introduce in the following section.

**Definition 7.** ($\alpha$-removable node, $\alpha$-critical) Given cohesive pattern $G = (V, E, D)$, $c \in V$. $c$ is called **$\alpha$-removable node** if $G - c$ is $\alpha$-dense. A cohesive pattern $G$ is called **$\alpha$-critical**, if every $\alpha$-removable node is a bridge node. The $\alpha$-removable nodes in an $\alpha$-critical graph are called **$\alpha$-critical nodes**.

The graph in Figure 2 is $\alpha$-critical, since the only $\alpha$-removable node is the bridge node $c$. Therefore, $c$ is called $\alpha$-critical node.

4 Algorithm

In this section, we introduce an algorithm, **cohesive pattern miner (CoPaM)**, for solving the cohesive pattern mining problem. This two phase algorithm adopts a level-wise bottom-up pattern enumeration, i.e. the search space of cohesive patterns of size $n$ is based on the cohesive patterns of size $n - 1$. We will analyze important properties of cohesive patterns in subsection 5 after introducing the algorithm in 4.2 and two non-integrated approaches in 4.1.

4.1 Cohesive Pattern Mining Baseline Approaches

A non-integrated cohesive pattern mining approach finds first all connected and dense patterns and checks for subspace cohesion afterwards. Alternatively, it first finds all subsets of nodes which are cohesive in a subspace and checks the density and connectivity constraints afterwards. In the experimental section we compare CoPaM with these two non-integrated baseline pattern miners, which we explain now in more detail:

**Baseline 1**

**Algorithm:** CoPaM-based Connected and Dense Graph Generation

**Postprocessing:** Checking against Subspace Cohesion Constraint

We generate all connected and dense graphs using the graph generation part from CoPaM, see Section 4.2. More specifically, we use a trivial subspace cohesion constraint which is true for any pattern and run CoPaM with it in order to generate all connected and dense graphs. As postprocessing we filter out all candidate patterns not satisfying the actual subspace constraint.

**Baseline 2**

**Algorithm:** Apriori-based Subspace Clustering

**Postprocessing:** Checking against Connectivity and Density Constraint

We generate all subsets of nodes (clusters) whose feature vectors satisfy the subspace constraint. We use an apriori-based approach which exploits the anti-monotonicity of the subspace cohesion function. From the identified clusters, we filter out the ones which do not fulfill the connectivity and density constraint.

For both baseline methods a maximality check is necessary.

A straight-forward baseline approach which considers graph data and feature vectors at the same time is to construct a second graph by thresholding the feature vector similarity and mining both graphs simultaneously. However, due to the curse of dimensionality similarity on the full space leads to significant loss of information. This is why in such applications subspace clustering methods have been proven to outperform full space clustering methods. However, this strategy of using two graphs cannot be adapted to subspace clustering.
4.2 CoPaM - Cohesive Pattern Miner

The pseudo code of our Cohesive Pattern Miner (CoPaM) for mining cohesive patterns can be found in Algorithm 1, the first phase, expand-by-one, in Algorithm 2 and the second phase, merge, in Algorithm 3.

The input of CoPaM is a feature graph $G$ and the following parameters: a density threshold $\alpha$, $\frac{1}{3} < \alpha \leq 1$, a subspace cohesion function $s$, a subspace cohesion threshold $\theta$, and a minimum number of dimensions $\theta_{\text{dim}}$. The output is the set of all maximal cohesive patterns.

The algorithm starts with a preprocessing phase, in which non-cohesive edges are removed from the input graph, since they can never be part of any cohesive pattern. In this reduced graph we identify all connected components. In the following each of these components is analyzed separately, applying the first and second phase of our algorithm.

Expand-by-one (Algorithm 2) takes as input cohesive patterns of size 2 and returns maximally expanded-by-one cohesive patterns. Let $\text{level}$ denote the number of nodes of the current cohesive patterns (initially it is 2). In each level, we expand existing cohesive patterns of size $\text{level}$ by any neighboring node obtaining patterns of size $\text{level} + 1$. Note that for every expanded pattern, the corresponding maximal cohesive feature subspace is uniquely determined. The expand-by-one method avoids redundant generation of candidate patterns as much as possible. If an expanded pattern $G + v$ fulfills the $\text{CP}$ constraint (Algorithm 2, line 10), then we know that $G$ was not maximal and we replace it by $G + v$ in the candidate set $\text{currCohesivePatterns}$, otherwise we add $G$ to the result set. After having considered all patterns of a certain level (size), we move to the next level until all patterns are maximally expanded-by-one (see Definition 4) - in the pseudo code this is reflected by implementing the variable $\text{currCohesivePatterns}$ as a queue. Therefore, expand-by-one does a breadth-first search. The advantage of this search strategy is that at any point of time we only need to keep cohesive patterns of two levels in memory - this reduces the amount of memory needed substantially.

The expand-by-one phase can generate only expanded-by-one cohesive patterns, which follows directly from its definition. We can show that if $\alpha \geq \frac{1}{2}$, all cohesive patterns are indeed expanded-by-one cohesive patterns. Therefore, the first phase finds all cohesive patterns for $\alpha \geq \frac{1}{2}$. If $\alpha$ is between $\frac{1}{3}$ and $\frac{1}{2}$ a second phase is required. In this second phase, called merge phase, the search space is restricted to merging candidates which were identified in the first phase. If $\alpha < \frac{1}{3}$ then the algorithm is not guaranteed to be complete. Note that in applications such as social network analysis and systems biology typically $\alpha$ is larger than $\frac{1}{3}$, in most case even larger than $\frac{1}{2}$.

Algorithm 1 CoPaM: Cohesive Pattern Miner

1: INPUT: $G = (V, E, D, A)$, $\alpha$, $s$, $\theta$, $\theta_{\text{dim}}$
2: OUTPUT: maximal cohesive patterns
3: PREPROCESSING: remove non-cohesive edges from $G$
4: for all connected components $C_i = (V_i, E_i)$ in $G$ do
5: $\text{currCohesivePatterns}_i \leftarrow \emptyset$
6: $\text{mergingCand}_i \leftarrow \emptyset$
7: FIRST PHASE: EXPAND-BY-ONE
8: for all edges $e = \{v_1, v_2\} \in E_i$ do
9: $G_e \leftarrow (\{v_1, v_2\}, \{\{v_1, v_2\}\}, D)$
10: $\text{currCohesivePatterns}_i.$add($G_e$)
11: $\text{currCohesivePatterns}_i$ $\leftarrow$ Expand-by-one($\text{currCohesivePatterns}_i$)
12: if $(\alpha < \frac{1}{3})$ then
13: SECOND PHASE: MERGE
14: $\text{mergedPatterns}$ $\leftarrow$ merge($\text{mergingCand}_i$)
15: $\text{currCohesivePatterns}_i.$add($\text{Expand-by-one}(\text{mergedPatterns})$)
16: $\text{currCohesivePatterns} = \cup \text{currCohesivePatterns}_i$
17: MAXIMALITY CHECK: remove non-maximal cohesive patterns from $\text{currCohesivePatterns}$
18: return $\text{currCohesivePatterns}$

Algorithm 2 First phase: Expand-by-one

1: INPUT: Queue $\text{currCohesivePatterns}$
2: OUTPUT: maximally expanded-by-one cohesive patterns,
merging candidates
3: Queue $\text{resultSet} \leftarrow \emptyset$
4: Set $\text{mergingCand} \leftarrow \emptyset$
5: while $(G \leftarrow \text{currCohesivePatterns}.\text{pop()} \neq \text{NULL})$ do
6: if isMergingCandidate($G$) then
7: $\text{mergingCand}.\text{add}(G)$
8: $G$.isMaximal $\leftarrow$ true
9: for all (neighboring nodes $v$ of $G$) do
10: if $(G + v$ fulfills $\text{CP}$ constraint) then
11: if NOT $\text{currCohesivePatterns}.\text{Contains}(G + v)$ then
12: $\text{currCohesivePatterns}.\text{add}(G + v)$
13: $G$.isMaximal $\leftarrow$ false
14: if $(G$.isMaximal$)$ then
15: $\text{resultSet}.\text{add}(G)$
16: return $\text{resultSet}$, $\text{mergingCand}$

The merge phase takes a set of merging candidates and joins two merging candidates if they have the same merging candidate node. This is efficiently supported by using a hashtable as index structure, whose key is the node id of the merging candidate node. As we show in subsection 5, we only need consider graphs which overlap by exactly one node, namely the merging
In this section, we first analyze the monotonicity properties of the CP constraint. We will see that the CP constraint is not anti-monotone - this makes it very different from other pattern mining algorithms which make heavy use of this property. In subsection 5.2 we show that the expand-by-one phase finds all cohesive patterns in this case. For \( \frac{3}{2} \leq \alpha < \frac{1}{2} \), the CP constraint is loose anti-monotone, but has some nice properties that allow to find all cohesive patterns by merging the patterns found in the expand-by-one phase. These properties and the correctness of the merge phase are discussed in 5.3.

5.1 Monotonicity properties Given that a graph \( G \) satisfies constraint \( C \), \( C \) is called anti-monotone if all subgraphs of \( G \) satisfy \( C \).

Let \( G \) be a graph \( G \) of size \( n \) satisfying constraint \( C \). \( C \) is called loose anti-monotone if there exists at least one subgraph of \( G \) of size \( n - 1 \) which fulfills \( C \) as well. The concept of loose anti-monotonicity was introduced by [3].

Frequent pattern mining algorithms, like the frequent item set mining algorithm a-priori [2], make use of the anti-monotonicity property of the support.

Unfortunately, \( CP \) constraint is not anti-monotone. As illustrated by the cohesive pattern in Figure 2, where the removal of \( e \) (the only \( \alpha \)-removable node) disconnects the graph, i.e. for a given graph not every node can be removed such that the remaining graph is still connected. However, in any graph, there exists a node, such that its removal does not disconnect the graph, see Lemma 5.1 below. This implies that the connectivity constraint is loose anti-monotone. In the following, we analyze the monotonicity properties of the density constraint and the simultaneous application of the density and connectivity constraint for different ranges of \( \alpha \). Note that the subpace cohesion constraint is anti-monotone by definition.

**Lemma 5.1.** Given a connected graph \( G = (V,E) \) of size at least 2. There exist two distinct nodes \( v_1,v_2 \in V \) such that \( G - v_1 \) and \( G - v_2 \) are connected.

**Proof.** We use induction on the number of nodes in \( G \). If \( G \) does not contain any bridge node, then we are done. Otherwise, let \( v \) be a bridge node in \( G \). Then \( G - v \) consists of \( l > 1 \) connected components \( G_1, \ldots, G_l \). If \( G_1 \) contains only one node, then this node is not a bridge node in \( G \). Suppose \( G_1 \) has more than one node. By induction hypothesis, there are two distinct nodes \( u \) and \( w \) in \( G_1 \) such that they are not bridge nodes in \( G_1 \). If \( vu \) or \( vw \), say \( vu \), is not an edge in \( E \), then \( u \) is not a bridge node in \( G \), therefore \( G - u \) is connected. Otherwise \( uv \) and \( vw \) are edges in \( G \), then \( G - u \) is connected. Thus, there is a node in \( G_1 \) which is not a bridge node.

A similar argument is used for \( G_2 \).

**Theorem 5.1.** The density constraint is loose anti-monotone.

**Proof.** Let \( G = (V,E) \) be an \( \alpha \)-dense graph, \( v \in V \), and \( G' = G - v = (V',E') \). We distinguish:
Let \( G \) be a cohesive pattern. We apply Theorem 5.4, containing an \( \alpha \)-removable node in \( G \) which is not a bridge node, i.e., \( G - v_n = G_{n-1} \) is a cohesive pattern. We apply this strategy recursively until \( j \), \( j \leq n \), contains an \( \alpha \)-critical node. This strategy corresponds to the second call of the expand-by-one phase (line 15 of Algorithm 1).

By Lemma 5.2, \( G_j \) contains exactly two connected components \( G_1 \) and \( G_2 \) and a set \( CC \) of connected \( \alpha \)-critical nodes, i.e., \( G_j = G_1 \cup CC \cup G_2 \). By Theorem, Decomposition Theorem, (see [1] for this lengthy and complicated proof), we know that there exists a partition, \( G_1 \) and \( G_2 \), \( G_1 \cup G_2 = CC \), \( |C_1 \cap C_2| \geq 1 \), of nodes in \( CC \) such that \( G_1 + 1 + G_2 + 2 \) are expanded-by-one and therefore found in the expand-by-one phase. \( G_1 + C_1 \) and \( G_2 + C_2 \) are merging candidates and will be merged into \( G_j \) in the merge phase.□

Algorithmic implications: The correctness proof is done in a top-down manner, i.e., for any cohesive pattern, we show that it can be decomposed in the described three components, two merging candidates and an expand-by-one part. However, our algorithm finds in a bottom-up manner all merging candidates and expands them afterwards. Since our algorithm follows exactly the proof, we have a guarantee that it finds all maximal cohesive patterns. In Figure 2, the
two merging candidates are $G_1 + c$ and $G_2 + c$. The component $CC$ consists only of node $c$.

6 Experiments

We evaluate CoPaM on three real-world datasets, one social network dataset and two biological datasets. Furthermore, we perform a runtime analysis on synthetic datasets and compare CoPaM with two baseline algorithms.

As other graph pattern mining algorithms [16], [24], CoPaM was evaluated in terms of efficiency and scalability on synthetic data sets. In addition we evaluate the meaningfulness of the cohesive patterns by providing anecdotic evidence on the social network data and by comparing the result to a given domain ontology in the biological dataset.

There exists no golden standard for the social network data or biological dataset. However in the biological domain, some background knowledge is available that is commonly used in bioinformatics to assess the quality of modules which correspond to our cohesive patterns.

Another difference between the social network and biological application is that in computational biology there exist already methods for finding modules as which our cohesive patterns can be interpreted. Existing methods in social network analysis find large communities which are not comparable to the relatively small patterns, e.g. as collaboration groups, which we find with CoPaM.

To assess the quality of the biological dataset, we compared it to two related state-of-the-art algorithms that operate on both graph data and feature vectors, MATISSE [20] and Co-clustering [11].

All experiments were performed on a PC running Linux with a 1.86GHz CPU and 4 GB of main memory.

6.1 Social Network Dataset

One of the applications of our algorithm is to identify collaboration groups. We used a co-authorship network which is based on the two well-known scientific literature digital databases citeseer\(^1\) and DBLP\(^2\). In [15], Newman showed that scientific collaboration networks reflect the properties of general social networks very well. We chose papers, written between 2000 and 2004, belonging to three different research areas: Theory, Machine Learning, and Databases & Data Mining. 1,942 authors were extracted out of these papers. We attached to each author the keywords (out of selected 603) which occurred in the abstracts of their papers. Therefore, our feature vectors were Boolean. We connected two authors via an edge if they co-authored at least one paper. In total, the dataset contains 4,919 edges.

We chose the following subspace cohesion function $s_{sn}$:

$$s_{sn}(V, D, true) = \forall v \in V, d \in D : F_d(v) = true$$

$F_d(v)$ denotes the feature value of node $v$ in dimension $d$. This subspace cohesion function requires that all members of a community have all keywords in subspace $D$ in common. Furthermore, we chose as threshold for the minimum dimensionality 16 and as density $\alpha = \frac{1}{3}$.

![Figure 3: Max. cohesive pattern from social network dataset](http://www.informatik.uni-trier.de/~ley/db/)

In total, CoPaM identified 59 collaboration groups of size 6 or larger. As anecdotic evidence of the meaningfulness of cohesive patterns we discuss the following examples:

1. Wei Wang, Philip Yu, Jiawei Han, Beng Ooi, Kian-Lee Tan, Hongjun Lu (density = 0.4)

2. Philip Yu, Jiawei Han, Charu Aggarwal, Laks Laskhamanan, Divesh Srivastava, H. Jagadish (density = 0.5)

Jiawei Han and Philip S. Yu are part of both patterns. Depending on the topic they have collaborated with different researchers. According to the first pattern they worked with Wei Wang, Beng Ooi, Kian-Lee

\(^1\)http://citeseer.ist.psu.edu/
\(^2\)http://www.informatik.uni-trier.de/~ley/db/
Tan and Hongjun Lu on statistical methods (some of the identified subspace dimensions are skew, mixture, and uniform). According to the second pattern, they worked with Charu Aggarwal, Laks Lakshmanan, Divesh Srivastava and H. Jagadish on hierarchical document mining (document, feature and hierarchical).

We also identified the cohesive pattern of size 18 which can be found in Figure 3. This pattern corresponds to the VLDB paper with the title The Propel Distributed Services Platform.

6.2 Biological datasets We used a human and a yeast datasets. In both datasets the nodes correspond to genes, the edges to interactions (protein-protein and genetic interactions) and the feature vectors are gene expression data. The identified maximal cohesive patterns have a specific biological meaning, namely modules.

Human Dataset (H. sapiens): The interaction network (graph data) was extracted from the BioGRID database [5], which integrates both protein-protein and genetic interactions from multiple publicly available datasets. For the expression data (feature vectors), we used the comprehensive human tissue expression dataset [7]. As suggested by the authors, we retained only variably expressed genes which showed at least 2-fold ratio variation from the mean in at least 2 experiments. The final dataset contains 3,628 nodes connected by 8,924 edges and 115 dimensions.

Yeast Dataset (S. cerevisiae): The network (graph) was downloaded from the BioGRID database [5]. The gene expression data (feature vectors) was acquired from [8], which contains fold changes of genes in 300 cDNA experiments. The final dataset contains 1,043 differentially expressed genes with 2,664 interactions and 300 dimensional feature vectors.

Due to the absence of comprehensive module annotations, testing for statistically significantly over-represented biological process gene ontology (GO) terms in a group of interest is the common method used for evaluation of module inference algorithms. We used the GoMiner tool3 for testing whether the maximal cohesive patterns are enriched with GO terms with P-values, that are corrected for multiple hypothesis testing, below a threshold of 0.01. We used three metrics to evaluate the quality of the results:

1. Enrichment (precision) is computed as the percentage of found modules that are enriched with at least one GO term.

2. Coverage (recall) is defined as the number of GO terms associated with an enriched cluster found by the method divided by the number of all GO terms in the dataset.

3. F-Value There usually is a trade-off between precision and recall. The F-Value captures this trade-off. Given enrichment $E$ and coverage $C$, it is computed as $F = \frac{2EC}{E + C}$.

The subspace cohesion function, $s_{bio}$ is defined as

$$s_{bio}(V, D, \theta_s) = \forall d \in D : \max \{ F_d(v), v \in V \} - \min \{ F_d(v), v \in V \} \leq \theta_s.$$ 

$F_d(v)$ denotes again the feature value of node $v$ in dimension $d$. This cohesion function requires that the expression values (or fold changes) of all genes (nodes) in a pattern induced by $V$ are within a range of $\theta_s$ in all experiments (dimensions) $D$. Note that if $F_d(v)$ refers to missing data, $s_{bio}$ is false.

For the comparison partners, we used the recommended parameter settings. For CoPaM, the density threshold was set to 0.65, based on the density distribution of known modules4. For yeast, we set 1.25 as the fold-change range threshold ($\theta_s$) with minimum dimensionality 140, which is also derived from the true modules. For the human dataset, we used more relaxed parameters of 1.4 and 10 for the fold-change range threshold ($\theta_s$) and minimum dimensionality respectively due to the high amount of noise and missing values in the human gene expression data.

Results: Table 1 and Table 2 show the results from the human and yeast datasets respectively. The best score for each category is identified with bold font. In both datasets, MATISSE and CoPaM consistently yield enrichment over 90%. However, coverage-wise we see that only Co-Clustering and CoPaM yield scores over 60% in the human dataset and CoPaM achieves the top coverage by a large margin in the yeast dataset. MATISSE

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Coverage</th>
<th>Enrichment</th>
<th>F-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co-Clustering</td>
<td>0.65</td>
<td>0.79</td>
<td>0.71</td>
</tr>
<tr>
<td>MATISSE</td>
<td>0.38</td>
<td>0.93</td>
<td>0.54</td>
</tr>
<tr>
<td>CoPaM</td>
<td>0.64</td>
<td>0.96</td>
<td>0.77</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Coverage</th>
<th>Enrichment</th>
<th>F-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co-Clustering</td>
<td>0.42</td>
<td>0.71</td>
<td>0.53</td>
</tr>
<tr>
<td>MATISSE</td>
<td>0.17</td>
<td>0.94</td>
<td>0.29</td>
</tr>
<tr>
<td>CoPaM</td>
<td>0.59</td>
<td>0.95</td>
<td>0.73</td>
</tr>
</tbody>
</table>

outputs only the statistically significant patterns, hence it achieves high enrichment. However, it performs poor in terms of coverage. On the other hand, Co-Clustering forces every node into a pattern, hence it yields high coverage at the cost of poor enrichment. Although CoPaM does not force every node to belong to a pattern, it achieves comparable, if not better, coverage, thanks to its completeness. This means that CoPaM is able to find patterns of a larger range of functionalities without sacrificing quality. This is supported by the F-value, in which CoPaM performs the best.

CoPaM finds connected, dense and homogeneous patterns, it allows overlap and does not force every node into a pattern. None of the comparison partners address all these issues simultaneously and we argue this is the main reason for the superiority of CoPaM. The runtime for the human dataset was 8 seconds and 18 seconds for the yeast dataset.

![Figure 4: Runtime for generating cohesive patterns of size 7 based on cohesive patterns of size 6.](image)

6.3 Synthetic Datasets In this section, we analyze the runtime of CoPaM on synthetic datasets. Generative graph models are a research area of its own. To our best knowledge, existing methods cannot take into account feature vectors. Generating the feature vectors independently of the graph data, makes the assumption that both data types are independent. However, people who are co-authors are interested in similar research areas and proteins which interact are more likely to have similar expression data [6]. In order to simulate real feature graphs as much as possible, we based our synthetic graphs on the social network dataset described in 5.1. We took the largest component after the removal of non-cohesive edges and made several copies of it, connecting the components randomly with cohesive edges. These graphs have the property that they are connected after the preprocessing phase in order to have a fair comparison. We chose the following parameter settings: \( \alpha = \frac{1}{3} \), \( s \) is the same as for the social network data set, minimum dimensionality is set to 20 which resulted in a largest component of size 746.

We generated graphs with the sizes of 746, 1492, 2238, 2984, 3730. The total runtimes and number of maximal cohesive patterns can be found in Table 3. The size of the largest pattern is 19. The runtime ranges from 50 seconds for the graph of size 746 to 560 seconds for the graph of size 3730. The number of maximal cohesive patterns is between 667 and 3419. Let us now have a closer look at the different levels of these runtimes. Recall that a level in our algorithm corresponds to the step of generating cohesive patterns of size \( n \) based on the ones of size \( n-1 \). If we examine, e.g. level 7, i.e. the time for generating cohesive patterns of size 7 based on the ones of size 6, we recognize a linear trend, see Figure 4. The data points in this figure correspond to the runtime for the different synthetically generated graphs. The runtime for generating 43,978 patterns in the graph of size 746 is 5 seconds and increases linearly with the number of cohesive patterns, up to 26 seconds for 238,489 patterns in the graph of size 3,730.

6.4 Comparison to baseline algorithms In this subsection, we compare the number of patterns produced by the baseline algorithms introduced in 4.1 (called Baseline 1 and 2) versus CoPaM. All these pattern mining algorithms are in each level (= size of pattern) output-sensitive algorithm, therefore the runtime is reflected by the number of patterns, as we have shown experimentally in the last subsection. We used again the dataset described in section 5.1 with the stated parameters. For the generation of patterns up to a size of 5 (a larger number caused a memory overflow), Baseline 1 generated 70 times more patterns than CoPaM and Baseline 2 11 times more. well with increased network size.

7 Conclusion While most existing methods for analyzing network data use graph data only, in many applications more informative graphs with feature vectors are given. Recently, integrated methods for mining graph and attribute data have emerged in several research communities. To mine patterns in such networks, we have introduced the novel problem of mining cohesive patterns, which combines the concepts of dense subgraphs and of subspace clus-
ters into a very powerful problem definition with important real life applications such as social network analysis and systems biology. The task is to find all maximal cohesive patterns, i.e. maximal dense and connected subgraphs with feature values that are homogeneous in a large enough subspace. The proposed CoPaM algorithm makes the computationally hard problem tractable by simultaneously pruning the search space based on the given density and subspace constraints. We prove that our algorithm is complete with respect to this problem definition. Our experiments on real life datasets show that the CoPaM algorithm produces meaningful patterns and outperforms existing integrated methods in terms of accuracy, and our evaluation on synthetic data demonstrates the scalability of CoPaM.

We conclude by discussing several directions for future research. First, we plan to investigate the parallelization of the CoPaM algorithm which promises to further improve its scalability to very large and high dimensional feature graphs. Secondly, we have experimented with two subspace cohesion functions that are appropriate for our driving applications. Other functions, such as a function based on the concept of order-preserving submatrices, that do not produce a unique maximal cohesive pattern for a given subgraph will require an extension of the CoPaM algorithm which is worth exploring as our current distant function cannot handle negative correlation, which is sometimes needed in biological applications.

References