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

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. 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 identifying small communities in social networks and functional modules in Protein-Protein interaction networks. We present the algorithm CoPaM (Cohesive Pattern Miner), 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 [15] and quasi-clique finding [14] 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 features of customers and expression data of genes. Often features and edges contain complementary information, i.e. neither the relationships can be derived from the feature vectors nor vice versa. In such scenarios the simultaneous use of both data types promises more meaningful and accurate results. Joint cluster analysis [3] aims at partitioning a graph with feature vectors into connected components whose nodes have similar feature vectors. [16] introduces a spectral clustering method which partitions graphs with feature vectors. While these approaches can exploit feature vectors 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. Furthermore a cohesive pattern 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. Moreover, 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 and 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 (cohesive patterns‘). Our algorithm effectively prunes the search space by simultaneously 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.

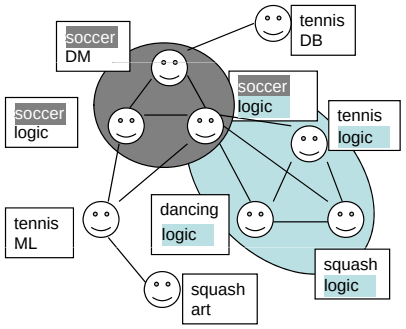

Figure 1: *Example of a social network of CS students*

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: Their members know each other quite well, 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 (A, B and C) consists of soccer players, united by their favorite sports, and the second one (G, H and I) 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 application 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. Two approaches, which successfully combined both data types, can be found in [20, 8].

1.2 Main contributions

- We introduce the novel problem of **mining cohesive patterns**, which integrates the concepts of finding dense subgraphs and subspace clustering.
- We develope the algorithm CoPaM (Cohesive Pattern Miner) that efficiently finds the set of all maximal cohesive patterns.
- We provide a theoretical analysis giving insights into the difficulty of the problem 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. 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. [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 α of all possible edges within the pattern. [14] and [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, [23] assumes a database of graphs with unique node identifiers, so that graph isomorphism is not an issue. The authors 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 algorithm, SPLAT, targets datasets containing mainly highly connected patterns. Relaxing the minimum support constraint, [9] 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., [4]). 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, [6] presents an algorithm based on a recursive application of shingling followed by a final clustering step. Graph partitioning algorithms such as normalized cut [15] can be considered as another approach for efficiently finding some of the densest subgraphs. Constrained on minimizing the cut size, these algorithms partition the graph into components. The small weight of the cut size is responsible for a higher density of the components.

A variety of approaches have been developed for integrated mining of graphs with associated feature vectors. Multi-relational clustering algorithms, e.g. the PRM-based approach of [19], partition a database of multiple related tables, which can be modeled as a multi-modal graph with feature vectors, into a specified number of clusters optimizing an objective function such as the likelihood. Joint cluster analysis [3] aims at partitioning a graph with feature vectors into connected components with similar feature vectors in all dimensions, a more specialized approach specifically targeting graphs with features. While these approaches take into account graphs with feature vectors, they do not ensure that the discovered clusters are dense and connected. The Co-Clustering method [8] defines an integrated distance function incorporating both the similarity of feature vectors 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 [20], a probabilistic method that determines connected subnetworks in graphs (such as interaction networks) that exhibit high feature (e.g., gene 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 vector graphs. First we give some definitions necessary to understand the problem definition.

Definition 1. (Feature vector graph) *A **feature vector graph** is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$, in which $\mathcal{V} = \{v_1, \ldots v_n\}$ denotes the node set and $\mathcal{E} \subseteq \{\{v_i, v_j\} \mid v_i, v_j \in \mathcal{V}, v_i \neq v_j\}$ the edge set. The function $\mathcal{F} : \mathcal{V} \rightarrow \mathcal{D}_1 \times \ldots \times \mathcal{D}_d$ is a **feature function** ($\mathcal{F}(v), v \in \mathcal{V}$ is a feature vector). The set $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_d\}$ is called the **feature space** of $\mathcal{G}$, $\mathcal{D}' \subseteq \mathcal{D}$ **feature subspace**.*

Social network data or protein interaction networks in combination with gene expression data are examples for feature vector graphs. For example, in Figure 1 the nodes in the network have a two-dimensional feature vector attached. This data can be formally represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$, where

$$\mathcal{V} = \{A, \ldots I\}, \mathcal{E} = \{\{A, B\}\{A, C\}, \ldots\},$$

$$\mathcal{D} = \{\{soccer, tennis, squash, dancing\}, \{DM, logic, ML, DB\}\}$$

and function $\mathcal{F}$, e.g. $\mathcal{F}(A) = (soccer, logic)$. Note that we denote with $\mathcal{G}$ the overall graph, and with G a subgraph in $\mathcal{G}$. In this paper we are interested in finding *cohesive patterns*, i.e. subgraphs G of $\mathcal{G}$ 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

$$s : \mathcal{P}(\mathcal{V}) \times \mathcal{P}(\mathcal{D}) \times \Re \rightarrow \{T, F\}$$

which has as input a subset V of the node set $\mathcal{V}$, a subset D of the feature space $\mathcal{D}$, and a real number, such that

$$(s(V, D, \theta_s) = T \wedge \nexists D' \supset D : s(V, D', \theta_s) = T) \Rightarrow$$

$$(s(V, D'', \theta_s) = T \Rightarrow D'' \subseteq D),$$

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. $s(V, D, \theta_s) = T \Rightarrow s(V', D', \theta_s) = T \; \forall V' \subset V, D' \subset D$

θ_s is called **subspace cohesion threshold**.

To illustrate the subspace cohesion function, consider again the example in Figure 1. All nodes in a community have the same value in at least one dimension. Formally, $s(V, D, \theta_s) = T$ if $\forall v \in V, d \in D : F_d(v) = c$ for some value c. So far, we have not restricted the size of D, i.e. in the worst case D is empty and s true for any node set V. 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 vector graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{F})$ and the following parameters:*

- *subspace cohesion function s,*
- *subspace cohesion threshold θ_s,*
- *dimensionality threshold θ_{dim} and*
- *density threshold α.*

*An induced subgraph $G = (V, E, D)$, $V \subset \mathcal{V}, E = \{v_1, v_2 | v_1, v_2 \in V, \{v_1, v_2\} \in \mathcal{E}\}, D \subset \mathcal{D}$, is called **cohesive pattern** if it satisfies the following three constraints:*

- ***Subspace cohesion constraint:** G is homogeneous in $D \subseteq \mathcal{D}$, i.e. $s(V, D, \theta_s) =$ true and $|D| \geq \theta_{dim} \geq 1$*
- ***Density constraint:** $d(G) := \frac{2|E|}{|V|(|V|-1)} \geq \alpha$. (In this case G is also called α-dense.)*
- ***Connectivity constraint:** G is connected.*

*We call the density constraint, the subspace cohesion 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**.*

We are particular interested in finding maximal cohesive patterns which are defined as follows:

DEFINITION 3. *[Maximal cohesive pattern] Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{D}, \mathcal{A})$ be a feature vector graph and $G = (V, E, D)$ be a cohesive pattern. G is called **maximal cohesive pattern**, if $\nexists V' \supset V$ and $\nexists D' \subseteq D$ such that the graph $G' = (V', E', D')$ induced by V' is also a cohesive pattern. Furthermore, we require D to be maximal, i.e. $\nexists D'' \subseteq \mathcal{D} \backslash D$, such that $G = (V, E, D'')$ is 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 $\mathcal{G}$ be a feature vector graph, s a subspace cohesion function, θ_s a subspace cohesion threshold, θ_{dim} a dimensionality threshold and α a density threshold, the **Cohesive Pattern Mining (CoPaM) Problem** is to find the set of all maximal cohesive patterns of $\mathcal{G}$ 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 [11]. 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 the worst case (all dense graphs are cohesive), our problem can be reduced to the problem of counting all cliques from a graph which is known to be #P-Complete [5]. Again this theoretical worst case does not typically 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 $\tau = (v_{i_1}, \ldots v_{i_n})$ 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 iteratively generated by starting from two connected nodes and adding one connected node at a time, such that all resulting patterns are cohesive.

Not all *maximally expanded-by-one* patterns are *maximal cohesive patterns*, as the example in Figure

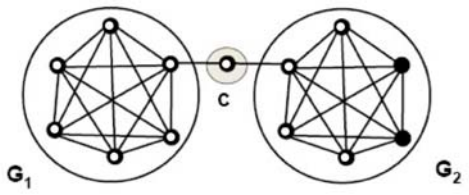

Figure 2: *Example for cohesive pattern containing α-critical node c for $\alpha = 0.41$. $G_1 + c$ and $G_2 + c$ are merging candidates.*

2 demonstrates. For $\alpha = 0.41$, the white nodes form a maximally expanded-by-one pattern, however this pattern is not maximal (the complete graph is maximal). Furthermore, the complete graph is also not expanded-by-one, since the only node which can be removed without violation of the density constraint is c. The node c is a bridge node which is defined in the following:

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. In Section 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*

$$\exists v \in V : deg(v) < deg(v') + 2 \forall v' \in V \setminus \{v\},$$

*where $deg(v)$ is the degree of v. Node 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. (α-REMOVABLE NODE, α-CRITICAL) *Given cohesive pattern $G = (V, E, D)$, $c \in V$. c is called α-**removable node** if $G - c$ is α-dense.*
*A cohesive pattern G is called α-**critical**, if every α-removable node is a bridge node. The α-removable nodes in an α-critical graph are called α-**critical nodes**.*

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

4 Algorithm

In this section, we introduce the algorithm **CoPaM (cohesive pattern miner)**, which solves 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 Section 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 pattern generation
Postprocessing: Checking against subspace cohesion constraint
First, we generate all connected and dense pattern using CoPaM, see Subsection 4.2. More specifically, we use a trivial subspace cohesion constraint which is true for any pattern and run CoPaM with this constraint in order to generate all connected and dense patterns. Second, as postprocessing, we filter out all candidate patterns not satisfying the subspace cohesion constraint.

Baseline 2
Algorithm: Apriori-based subspace clustering
Postprocessing: Checking against connectivity and density constraints
First, we generate all subsets of nodes (clusters) whose feature vectors satisfy the subspace cohesion constraint. We use an apriori-based approach which exploits the anti-monotonicity of the subspace cohesion function to do that. Second, from the identified clusters, we filter out the ones which do not fulfill the connectivity and density constraints.

For both baseline methods a maximality check is necessary.

One strategy commonly used which could be considered as a third baseline is the following. We construct a second graph by thresholding the feature vector similarity and mine both graphs simultaneously. However, similarity on the complete feature space leads to significant loss of information due to the curse of dimensionality. This is why, in such applications, subspace cluster-

ing 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 the Cohesive Pattern Miner (CoPaM) 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 vector graph $\mathcal{G}$ and the following parameters: a density threshold α, $\frac{1}{3} < \alpha \leq 1$, a subspace cohesion function s, a subspace cohesion threshold θ_s and a minimum number of dimensions θ_{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 *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 *level* by any neighboring node obtaining patterns of size $level + 1$. 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 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 $currCohesivePatterns$, 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. This is reflected in the pseudo code by implementing the variable $currCohesivePatterns$ 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 generates only expanded-by-one cohesive patterns, which follows directly from its definition. We will show in the next section 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 α 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 α 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), α, s, θ_s, θ_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:    currCohesivePatterns_i ← ∅
6:    mergingCand_i ← ∅
7:    FIRST PHASE: EXPAND-BY-ONE
8:    for all (edges e = {v_1, v_2} ∈ E_i) do
9:       G_e ← ({v_1, v_2}, {{v_1, v_2}}, D)
10:      currCohesivePatterns_i.add(G_e)
11:   currCohesivePatterns_i ←
      Expand-by-one(currCohesivePatterns_i)
12:   if (α < 1/2) then
13:      SECOND PHASE: MERGE
14:      mergedPatterns_i ← merge(mergingCand_i)
15:      currCohesivePatterns_i.
         add(Expand-by-one(mergedPatterns_i))
16: currCohesivePatterns = ∪_i currCohesivePatterns_i
17: MAXIMALITY CHECK: remove non-maximal cohesive
    patterns from currCohesivePatterns
18: return currCohesivePatterns
```

Algorithm 2 First phase: Expand-by-one

```
1: INPUT: Queue currCohesivePatterns
2: OUTPUT: maximally expanded-by-one cohesive pat-
   terns,
   merging candidates
3: Queue resultSet ← ∅
4: Set mergingCand ← ∅
5: while (G ← currCohesivePatterns.pop() ≠ NULL)
   do
6:    if (α < 1/2) then
7:       if (isMergingCandidate(G)) then
8:          mergingCand.add(G)
9:       G.isMaximal ← true
10:   for all (neighboring nodes v of G) do
11:      if (G + v fulfills CP constraint) then
12:         if NOT currCohesivePatterns.Contains(G+v)
            then
13:            currCohesivePatterns.add(G + v)
14:            G.isMaximal ← false
15:   if (G.isMaximal) then
16:      resultSet.add(G)
17: return resultSet, mergingCand
```

Algorithm 3 Second Phase: Merge

```
1: INPUT: hashtable mergingCand
2: OUTPUT: α-critical graphs
3: result ← ∅
4: for all (G1, G2 ∈ mergingCand which have only the
   merging cand. node in common) do
5:    if (( G1 ∪ G2) is α-critical) then
6:       result.add(G1 ∪ G2)
7: return result
```

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 keys are the node ids of the merging candidate node. As we show in Section 5, we only need consider graphs which overlap by *exactly one* node, namely the merging candidate node. We return only α-critical cohesive patterns. We have to apply again the expand-by-one phase on these graphs.

The goal of the final **maximality check** is to remove cohesive patterns which are not maximal. This postprocessing step is necessary, since the CP constraint is not anti-monotone, see Section 5. This is different from frequent itemset mining, where the anti-monotonicity property guarantees that we find only maximal patterns. The result set of CoPaM is implemented as a queue, such that any new element is added at the beginning and during this maximality check we can retrieve the elements in reverse order of their insertion in constant time. Furthermore, the maximality check makes use of a hashtable and an array. The keys of the hashtable are the node ids. In the buckets we store all maximal cohesive patterns which contain this particular node. In a second data structure, an array, we record the number of elements in each bucket. We insert all cohesive patterns identified by CoPaM into the hashtable in opposite order of their creation, i.e. starting with the largest pattern. When adding a new pattern P we do the following:

- Find smallest bucket $B_{smallest}$ to which P has to be added, using the array information.
- Test whether P is subset of any of the elements in $B_{smallest}$.
  If P is not subset, add P to the hashtable.
  If P is subset, then discard it.

This concludes the description of CoPaM. In the following section we prove its correctness for $\alpha \geq \frac{1}{3}$.

5 Correctness of CoPaM

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 heavily use of this property. In subsection 5.2, we show that the CP constraint is loose anti-monotone (defined in 5.1) for $\frac{1}{2} \leq \alpha$, and we prove that the expand-by-one phase finds all cohesive patterns in this case. For $\frac{1}{3} \leq \alpha < \frac{1}{2}$, the CP constraint is not loose anti-monotone, but it has some nice properties that allow for finding 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 [2].

Frequent pattern mining algorithms, like the frequent item set mining algorithm a-priori [1], make use of the anti-monotonicity property of the support. Unfortunately, CP **constraint is not anti-monotone**. For a given graph not every node can be removed such that the remaining graph is still connected. A counter example is shown by the cohesive pattern in Figure 2, where the removal of c (the only α-removable node) disconnects the graph. However, in any graph, there exists a node, such that its removal does not disconnect the graph, see Lemma 5.1. 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 α. Note that the subspace 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 wv 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 α-dense graph, $v \in V$, and $G'=G-v=(V',E')$. We distinguish:

- $\exists v \in V$: $deg_G(v) < \lceil \alpha(|V|-1) \rceil$.
  $\Rightarrow |E'| = |E| - d_G(v) \geq \alpha\frac{(|V|-1)(|V|-2)}{2} = \alpha\frac{(|V'|)(|V'|-1)}{2}$
- $\forall v \in V : deg_G(v) \geq \lceil \alpha(|V|-1) \rceil$
  Let $v \in V$ be the node with minimum degree $m \geq \lceil \alpha(|V|-1) \rceil$ in G.
  $\Rightarrow |E'| \geq \frac{|V|m}{2} - m \geq \frac{1}{2}(|V|-2)m \geq \frac{1}{2}(|V|-2)\alpha(|V|-1) = \alpha\frac{|V'|(|V'|-1)}{2}$

Therefore, G' is α-dense. □

To summarize the results of this subsection:

- The subspace cohesion constraint is anti-monotone.
- The density constraint is loose anti-monotone.
- The connectivity constraint is loose anti-monotone.

5.2 Correctness of CoPaM for $\frac{1}{2} \leq \alpha$. After having analyzed the monotonicity properties of the constraints separately, we will now analyze the simultaneous satisfaction of them.

THEOREM 5.2. *Simultaneous satisfaction of the connectedness and density constraint is loose anti-monotone for $\alpha \geq \frac{1}{2}$.*

Proof. Let $G=(V,E,D)$ be connected and α-dense for $\alpha \geq \frac{1}{2}$. Assume the only α-removable nodes in G are bridge nodes. Let $b \in V$ be one of them. Then $G-b$ contains at least two connected components G_1 and G_2. There is at least one node in G_1, say v_1, whose removal does not disconnect G (application of Lemma 5.1 to G_1+v). Since v_1 was not α-removable, $deg_G(v_1) > \alpha(|V|-1)$. Therefore, G_1 contains more than $\alpha(|V|-1)-1+1 \geq \frac{1}{2}(|V|-1)$ nodes. Using a similar argument for G_2, results in G having more than $|V|$ nodes, which is a contradiction. Therefore, there exists a node $v \in V$, such that $G-v$ is connected and α-dense. □

Algorithmic implication of Theorem 5.2: Let $\alpha \geq \frac{1}{2}$. Let's say, we want to find a cohesive pattern $G=(V,E)$. By Theorem 5.2, there exists a node $v \in V$ such that $G-v$ is α-dense and connected. Since the subspace cohesion constraint is anti-monotone by definition, we know that $G-v$ is again a cohesive pattern. If we apply this top-down strategy recursively, we will end up with a cohesive edge. The set of all cohesive edges is exactly the input to CoPaM. Within the expand-by-one phase, we expand current cohesive patterns by neighboring nodes. Therefore, the expand-by-one phase finds all cohesive patterns for $\alpha \geq \frac{1}{2}$.

THEOREM 5.3. *The connectedness and α density constraints together are **not** loose anti-monotone, if $\alpha < \frac{1}{2}$.*

Example: Figure 2 shows a cohesive pattern for $\alpha = 0.41$. The only 0.41-removable node is c. Since c is a bridge node, its removal disconnects the graph, and therefore the connectedness constraint is violated. □

5.3 Correctness of CoPaM for $\frac{1}{3} \leq \alpha < \frac{1}{2}$. We have seen that the CP constraint is not loose anti-monotone for $\frac{1}{2} < \alpha$. We still can guarantee the correctness of CoPaM by applying the second phase, merging phase, as we will see in the following.

LEMMA 5.2. *Let $\frac{1}{3} \leq \alpha < \frac{1}{2}$ and $G=(V,E,D)$ be a cohesive pattern. The removal of all α-removable nodes of V disconnects G into two connected subgraphs.*

Proof. The proof is similar to the one of Theorem 5.2 using $\alpha = \frac{1}{3}$ instead of $\frac{1}{2}$ and deriving a contradiction that it is not possible to have three subgraphs. For details, see our extended version [12]. □

THEOREM 5.4. *If $\frac{1}{3} \leq \alpha < \frac{1}{2}$, a cohesive pattern G is either expanded-by-one, or it can be decomposed into an expand-by-one part and two merging candidates which are expanded-by-one.*

Proof. Let G be a cohesive pattern of size n. If G is expanded-by-one, we use the same argument as in Theorem 5.2. Otherwise, let $G_n = G$ be a cohesive pattern which is not expanded-by-one. Let v_n be an α-removable node in G_n which is not a bridge node, i.e. $G-v_n = G_{n-1}$ is a cohesive pattern. We apply this strategy recursively until G_j, $j \leq n$, contains an α-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 α-critical nodes, i.e. $G_j = G_1 \cup CC \cup G_2$. By Theorem, *Decomposition Theorem*, (see [12] for this lengthy and complicated proof), we know that there exists a partition, C_1 and C_2, $C_1 \cup C_2 = CC$, $|C_1 \cap C_2| \geq 1$, of nodes in CC such that $G_1 + C_1$ and $G_2 + C_2$ 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. We also perform a runtime analysis on synthetic datasets and compare CoPaM with two baseline algorithms.

As other graph pattern mining algorithms [14], [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 anecdotal 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.

In computational biology, there already exist methods for finding modules as which our cohesive patterns can be interpreted. However, existing methods in social network analysis find large communities which are not comparable to the relatively small patterns (e.g. collaboration groups) that 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 [8].

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 [13], 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 $\alpha = \frac{1}{3}$.

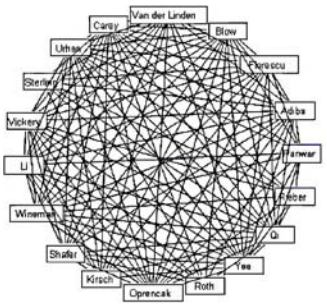

Figure 3: *Max. cohesive pattern from social network dataset*

In total, CoPaM identified 59 collaboration groups of size 6 or larger. As anecdotal 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

[1] http://citeseer.ist.psu.edu/
[2] http://www.informatik.uni-trier.de/~ley/db/

with different researchers. According to the first pattern they worked with Wei Wang, Beng Ooi, Kian-Lee 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. Our two biological dataset are human and yeast. In both datasets the nodes correspond to genes, the edges to interactions (protein-protein and genetic interactions) and the feature vectors to gene expression data. It has been argued, e.g. by [7, 20, 8], that the combined analysis of these two data types for the identification of modules is more promising than the individual analysis. Indeed, the identified maximal cohesive patterns have a specific biological meaning, namely modules, as we will show in the following.

Human Dataset (*H.sapiens*): The interaction network (graph data) was extracted from the BioGRID database [18], 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 [17]. 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 [18]. The gene expression data (feature vectors) was acquired from [10], 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, a common method for evaluating module inference algorithms is testing for statistically significant over-represented biological process gene ontology (GO) terms in the group of interest. We used the GoMiner tool[3] for testing whether the maximal cohesive patterns are enriched with GO terms with P-values, which are corrected for multiple hypothesis testing, below a threshold of 0.01. We used three metrics to evaluate the quality of the results:

Table 1: Quality assessment on the human dataset

Algorithm	Coverage	Enrichment	F-Value
Co-Clustering	**0.65**	0.79	0.71
MATISSE	0.38	0.93	0.54
CoPaM	0.64	**0.96**	**0.77**

Table 2: Quality assessment on the yeast dataset

Algorithm	Coverage	Enrichment	F-Value
Co-Clustering	0.42	0.71	0.53
MATISSE	0.17	0.94	0.29
CoPaM	**0.59**	**0.95**	**0.73**

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* The *F-Value* captures the trade-off between precision and recall. 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 θ_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 modules[4]. For yeast, we set 1.25 as the fold-change range threshold (θ_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 (θ_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

[3] http://discover.nci.nih.gov/gominer/

[4] http://www.yeastgenome.org

Table 3: Runtime for syn. generated graphs

Nr. of nodes	746	1,492	2,238	2,984	3,730
Runtime in sec	50	111	306	421	560
♯ max. cp.	667	1347	2037	2723	3419

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 outputs only the statistically significant patterns, hence it achieves high enrichment. However, it performs poorly 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, due to its completeness (for the used values of α). 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.

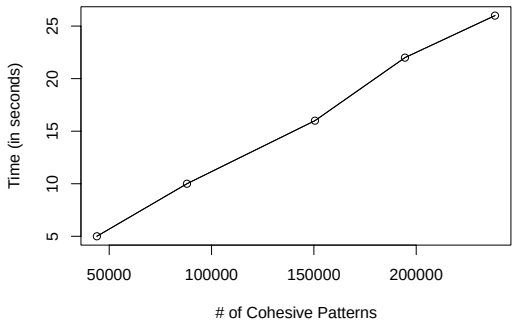

Figure 4: *Runtime for generating cohesive patterns of size 7 based on cohesive patterns of size 6.*

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 [7]. In order to simulate real feature vector 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. For example, on level 7, where the time for generating cohesive patterns of size 7 based on the ones of size 6 is measured, 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. We compare the number of patterns produced by the baseline algorithms introduced in 4.1 (called Baseline 1 and 2) versus CoPaM. All algorithms are output-sensitive algorithms in each level (=size of patterns), therefore the runtime is reflected by the number of patterns. We also have shown this experimentally in the previous 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 eleven times more.

7 Conclusion

While most existing methods for analyzing network data use graph data only, in many applications more infor-

mative graphs with feature vectors are given. Recently, integrated methods for mining graph data and feature vectors 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 clusters 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. 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 cohesion constraints. We prove that our algorithm is complete with respect to this problem definition for a density threshold above $\frac{1}{3}$. Our experiments on real life datasets show that CoPaM produces meaningful patterns. Our evaluation on synthetic data demonstrates the scalability of our algorithm.

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 vector graphs. Second, so far we have experimented with two subspace cohesion functions that are appropriate for our driving applications. In the future, we plan to use functions that do not produce a unique maximal cohesive pattern for a given subgraph, e.g. functions based on the concept of order-preserving submatrices. This extension of the CoPaM algorithm is worth exploring especially in the context of biological applications.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. of COMAD*, 1993.

[2] F. Bonchi and C. Lucchese. Pushing tougher constraints in frequent pattern mining. In *Proc. of PAKDD*, 2005.

[3] M. Ester, R. Ge, B. J. Gao, Z. Hu, and B. Ben-Moshe. Joint cluster analysis of attribute data and relationship data: the connected k-center problem. In *Proc. of SDM*, 2006.

[4] U. Feige, D. Peleg, and G. Kortsarz. The dense k - subgraph problem. *Algorithmica*, 2001.

[5] D. Johnson M. Garey. *Computer and Intractability: a Guide to the theory of NP-Completeness*. 1979.

[6] D. Gibson, R. Kumar, and A. Tomkins. Discovering large dense subgraphs in massive graphs. In *Proc. of VLDB*, 2005.

[7] A. Grioriev et al. A relationship between gene expression and protein interactions on the proteome scale: analysis of the bacteriophage t7 and the yeast saccharomyces cerevisiae. *Nucleic Acids Research*, 2001.

[8] D. Hanisch, A. Zien, R. Zimmer, and T. Lengauer. Co-clustering of biological networks and gene expression data. *Bioinformatics*, 2002.

[9] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 2005.

[10] T. Hughes et al. Functional discovery via a compendium of expression profiles. *Cell*, July 2000.

[11] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, 1972.

[12] F. Moser, R. Colak, A. Rafieyh, M. Ester. Mining Cohesive Patterns from Graphs with Feature Vectors - Extended Version. http:/www.cs.sfu.ca/∼ester/papers/SDM-2009-CohesivePatterns_ExtendedVersion.pdf

[13] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. In *Physical Review*, 2004.

[14] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *Proc. of ACM SIGKDD*, 2005.

[15] J. Shi and J. Malik. Normalized cuts and image segmentation. In *Proc. of IEEE TPAMI*, 2000.

[16] M. Shiga, I. Takigawa, and K. Mamitsuka. A spectral clustering approach to optimally combining numberical vectors with a modular network. In *Proc. of ACM SIGKDD*, 2007.

[17] R. Shyamsundar et al. A dna microarray survey of gene expression in normal human tissues. *Genome Biology*, 2005.

[18] C. Stark et al. Biogrid: a general repository for interaction datasets. *Nucleic Acids Research*, (Database issue), 2006.

[19] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proc. of UAT*, 2002.

[20] I. Ulitsky and R. Shamir. Identification of functional modules using network topology and high-throughput data. *BMC Systems Biology*, (8), 2005.

[21] J. Wang, Z. Zeng, and L. Zhou. Clan: An algorithm for mining closed cliques from large dense graph databases. In *Proc. of ICDE*, 2006.

[22] S. Wasserman and K. Faust. *Social Network Analysis*. Cambridge Univ. Press, 1994.

[23] X. Yan, X. J. Zhou, and J. Han. Mining closed relational graphs with connectivity constraints. In *Proc. of ACM SIGKDD*, 2005.

[24] Z. Zeng, J. Wang, L. Zhou, and G. Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In *Proc. of ACM SIGKDD*, 2006.

Acknowledgment We would like to thank Fereydoun Hormozdiari for his valuable contribution in the early discussion phase.