Iterative Subgraph Mining for Principal Component Analysis

Submitted for Blind Review

Abstract

Graph mining methods enumerate frequent subgraphs efficiently, but they are not necessarily good features for machine learning due to high correlation among features. Thus it makes sense to perform principal component analysis to reduce the dimensionality and create decorrelated features. We present a novel iterative mining algorithm that captures informative patterns corresponding to major entries of top principal components. It repeatedly calls weighted substructure mining where example weights are updated in each iteration. The Lanczos algorithm, a standard algorithm of eigendecomposition, is employed to update the weights. In experiments, our patterns are shown to approximate the principal components obtained by frequent mining.

1. Introduction

Graphs are general and powerful data structures that can be used to represent diverse kinds of objects. Much of the real world data is represented not as vectors, but as graphs including sequences and trees, for example, biological sequences, semi-structured texts such as HTML and XML, chemical compounds, RNA secondary structures, and so forth. To analyze graph databases, graph mining methods such as AGM [5], gSpan [20] or Gaston [10] have been successfully applied. They enumerate all frequent subgraphs (i.e., patterns) whose frequency is above a minimum support threshold. After the mining, we can create a feature space using the patterns as in Figure 1. Here, the existence of a pattern is represented as 1, otherwise -1.

Since the dimensionality of this feature space is typically too high to browse comfortably, it is required to map the feature space to a low dimensional space. Principal component analysis [15] is the most popular method of dimensionality reduction. Principal components are computed as the eigenvectors of the covariance matrix. When the number of examples is smaller than the dimensionality, it is also possible to calculate them from the Gram matrix [15]. It is commonly practised to map high dimensional objects to a two or three dimensional space to visualize the relationship between them. PCA is also used for indexing graphs by taking the patterns corresponding to major elements of principal components. Such a technique is known as latent semantic indexing (LSI) in the text processing community [3, 2]. Here, each principal component represents a topic, and its major elements correspond to the set of words describing the topic. Also in our graph cases, principal components are interpretable based on the patterns associated to them.

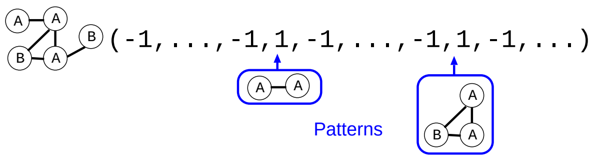

Figure 1. Feature space based on subgraph patterns. The feature vector consists of binary pattern indicators.

However, the following naive approach has efficiency problems;

1. First, frequent subgraph mining is applied to create the feature space.

2. Then, principal component analysis is performed.

It is because we need to lower the minimum support threshold to capture detailed information. We present an iterative mining approach for PCA, where salient patterns are progressively collected by several distinct graph mining calls. In each mining, real-valued weights are assigned to transactions (graphs), and the patterns satisfying the weighted support criterion is enumerated (i.e., weighted substructure mining). Such iterative mining approaches have been applied to a wide variety of problems, including boosting [7, 13], LARS [17], novelty detection [11], partial least squares regression [14], and clustering[18, 19]. In comparison to the naive approach, these algorithms are shown to achieve better efficiency without losing prediction accuracy.

For creating an iterative mining method for PCA, we need to take a close look at the numerical methods of eigendecomposition [1, 16, 4]. We particularly focus on the

Lanczos method [8], which is the most popular and widely used in numerical software including MATLAB. One of the key steps of the Lanczos method is matrix-vector multiplication between the design matrix and a Lanczos vector. In our algorithm, called graph PCA (gPCA), this step is approximated using weighted subgraph mining. In experiments, we evaluate gPCA in terms of approximation error and computational time. For interpretability, the number of patterns should be minimized while keeping the approximation error small. We will show that gPCA's patterns can approximate the principal components.

The rest of the paper is organized as follows. In Section 2, we give basic notations and define our problems. Section 3 reviews the Lanczos method and how it is used for principal component analysis. In Section 4, the Lanczos algorithm is applied to graph data. Section 5 reviews the weighted mining method. Section 6 contains our experiments evaluating the approximation accuracy and efficiency of gPCA. Section 7 concludes the paper.

2. Preliminaries

We deal with undirected, labeled and connected graphs. To be more precise, we define the graph and its subgraph as follows:

Definition 1 (Labeled connected graph) *A labeled graph is represented in a 4-tuple $G = (V, E, \mathcal{L}, l)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, $\mathcal{L}$ is a set of labels, and $l : V \cup E \to \mathcal{L}$ is a mapping that assigns labels to the vertices and edges. A labeled connected graph is a labeled graph such that there is a path between any pair of vertices.*

Definition 2 (Subgraph) *Let $G' = (V', E', \mathcal{L}', l')$ and $G = (V, E, \mathcal{L}, l)$ be labeled connected graphs. G' is a subgraph of G ($G' \subseteq G$) if the following conditions are satisfied: (1) $V' \subseteq V$, (2) $E' \subseteq E$, (3) $\mathcal{L}' \subseteq \mathcal{L}$, (4) $\forall v' \subseteq V', l(v') = l'(v')$ and (5) $\forall e' \subseteq E', l(e') = l'(e')$. If G' is a subgraph of G, then G is a supergraph of G'.*

Let p be a subgraph pattern in a graph, and $\mathcal{P}$ be the set of all patterns, i.e., the set of all subgraphs included in at least one graph. Given n graphs in the database, the feature vector of each graph G_i is a d-dimensional vector $\boldsymbol{x}_i$, where a component corresponding to pattern p is written as

$$x_{ip} = \begin{cases} 1 & \text{if } p \subseteq G_i, \\ -1 & \text{otherwise} \end{cases}$$

Denote by X the design matrix whose i-th row corresponds to $\boldsymbol{x}_i$. In this paper, principal components $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m \in \Re^d$ corresponds to m major eigenvalues of the covariance matrix $X^\top X$. Here, we do not centralize the design matrix X as in latent semantic indexing [3]. Principal components can also be computed from the Gram matrix $A = XX^\top$ [15]. The eigenvalues of A are equal to those of $X^\top X$. If the major eigenvectors of A are denoted as $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$, then the principal components are recovered as

$$\boldsymbol{z}_i = X^\top \boldsymbol{v}_i. \tag{1}$$

Our algorithm follows the latter path, namely, computing the eigenvectors of the Gram matrix. However, the exact computation of the Gram matrix and the recovery of principal components (1) requires the whole feature space. In our case, it is intractable, because all the patterns have to be enumerated a priori.

Therefore, we need to create a restricted design matrix $\tilde{X}$ where a tractable number of patterns are used. The selection of patterns is of utmost importance in obtaining accurate approximations $\tilde{\boldsymbol{z}}_i$. One obvious way to reduce the number of patterns is to use the minimum support threshold, and create a feature space with frequent patterns only. However, as shown in experiments later, it does not lead to accurate approximation due to high correlation among features. We propose an iterative mining method based on the Lanczos method for calculating major eigenvectors. The weighted subgraph mining is embedded to the Lanczos iteration. A handful of patterns are mined with different criteria in each iteration, thereby avoiding the correlation problem effectively.

3. The Lanczos algorithm

In this section, we review the Lanczos algorithm briefly. Algorithm 1 shows how to compute the all eigenvalues and eigenvectors of a $n \times n$ symmetric matrix A. Let us denote the eigenvalues of A as $(\lambda_1, \ldots, \lambda_n)$, and denote by V the matrix whose i-th column corresponds to the i-th eigenvector. Let $Q = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n)$ denote an orthogonal matrix. Then, the following transformation does not alter the eigenvalues

$$T = Q^\top A Q. \tag{2}$$

The Lanczos algorithm finds a matrix Q such that T is tridiagonal. Here, T is parametrized as follows,

$$\boldsymbol{T} = \begin{bmatrix} \alpha_1 & \beta_1 & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \beta_{n-1} \\ 0 & \ldots & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

The Lanczos algorithm computes the entries of T and Q progressively using elementary matrix computations. The columns of Q are called "Lanczos vectors". Since T is tridiagonal, the eigendecomposition can be efficiently done via

Schur decomposition. Denote by R the eigenvectors of T. Then, the eigenvectors of A are obtained as

$$V = QR. \tag{3}$$

This algorithm is employed in many numerical computation software including MATLAB. The orthogonal matrix leading to tridiagonal T is not unique. One can start from an arbitrary initial vector $\boldsymbol{q}_1$, and always obtain a tridiagonal matrix as T.

Algorithm 1 The Lanczos algorithm.

```
1: Input: $\boldsymbol{A} \in \mathbb{R}^{n\times n}, \boldsymbol{q}_1 \in \mathbb{R}^n, \|\boldsymbol{q}_1\| = 1.$
2: Output: All eigenpairs $[\boldsymbol{\lambda}, V]$
3: Initial: $\boldsymbol{r}_0 = \boldsymbol{q}_1;\ \beta_0 = 1;\ k = 0$
4: while $\beta_k \neq 0$ do
5:     $\boldsymbol{q}_k = \boldsymbol{r}_k/\beta_k$
6:     $k = k + 1$
7:     $\alpha_k = \boldsymbol{q}_k^\top \boldsymbol{A}\boldsymbol{q}_k$
8:     $\boldsymbol{r}_k = \boldsymbol{A}\boldsymbol{q}_k - \beta_{k-1}\boldsymbol{q}_{k-1} - \alpha_k\boldsymbol{q}_k$
9:     $\beta_k = \|\boldsymbol{r}_k\|_2$
10: end while
11: $[\boldsymbol{\lambda}, R] = EigenDecomposition(\boldsymbol{T})$
12: $V = QR$
```

In principal component analysis, we need only top m eigenpairs. In that case, we can stop the Lanczos algorithm after $K(< n)$ steps. Denote by $T_k, Q_k, R_k, V_k, \boldsymbol{\lambda}_k$ the early versions of $T, Q, R, V, \boldsymbol{\lambda}$ after k iterations, respectively. The eigenpairs $(\boldsymbol{\lambda}_k, V_k)$ are only approximation of top-k eigenpairs of A. However, there is an important property that the eigenvectors corresponding to larger eigenvalues are approximated more accurately. Thus, if k is sufficiently larger than m, the top-m eigenvectors in V_k are accurate estimates of the top-m eigenvectors of A. For rigorous error analysis, see [12]. An interesting point is that the accuracy depends on the initial vector $\boldsymbol{q}_1$. If $\boldsymbol{q}_1$ is set to the first eigenvector of A, V_k gives exactly the top-k eigenvectors [6]. However, since we do not know a priori the first eigenvector, $\boldsymbol{q}_1$ is usually set to an arbitrary vector. In our experiments, we always used $\boldsymbol{q}_1 = \boldsymbol{1}/\sqrt{n}$.

Now the question is how to determine when to stop the iteration. Here, the iteration is terminated, if all the m eigenvalues have converged [16, 4]:

$$\frac{\|\boldsymbol{A}\boldsymbol{V}_i - \lambda_i \boldsymbol{V}_i\|_2}{\lambda_i} = \frac{|\beta_k||R_{ki}|}{\lambda_i} < \sigma, \quad \forall i = 1, \dots, m, \tag{4}$$

where σ is the parameter controlling error tolerance. We set $\sigma = 0.01$ in our experiments.

It is known that the Lanczos algorithm is prone to rounding errors and the orthogonality between the Lanczos vectors is quickly lost. Several methods have been proposed for this problem, such as implicitly/explicitly restarted Lanczos and selective/partial orthogonalization [16]. In this paper, we apply the Gram-Schmidt procedure to ensure the orthogonality between the Lanczos vectors. Algorithm 2 summarizes the Lanczos algorithm with early stopping and reorthogonalization.

Algorithm 2 The Lanczos algorithm with early stopping and reorthogonalization.

```
1: Input: $\boldsymbol{A} \in \mathbb{R}^{n\times n}, \boldsymbol{q}_1 \in \mathbb{R}^n$, $\|\boldsymbol{q}_1\| = 1, m$
2: Output: Top eigenpairs $\{(\lambda_i, \boldsymbol{v}_i)\}_{i=1}^m$
3: Initial: $\boldsymbol{r}_0 = \boldsymbol{q}_1;\ \beta_0 = 1;\ k = 0$
4: while (4) is not true do
5:     $\boldsymbol{q}_k = \boldsymbol{r}_k/\beta_k$
6:     $k = k + 1$
7:     $\alpha_k = \boldsymbol{q}_k^\top \boldsymbol{A}\boldsymbol{q}_k$
8:     $\boldsymbol{r}_k = \boldsymbol{A}\boldsymbol{q}_k - \beta_{k-1}\boldsymbol{q}_{k-1} - \alpha_k\boldsymbol{q}_k$
9:     for $j = 1 : k - 1$ do                ▷ Reorthogonalization
10:        $\boldsymbol{r}_k = \boldsymbol{r}_k - \boldsymbol{q}_j(\boldsymbol{q}_j^\top \boldsymbol{r}_k)$
11:    end for
12:    $\beta_k = \|\boldsymbol{r}_k\|$
13:    $[\boldsymbol{\lambda}_k, R_k] = EigenDecomposition(\boldsymbol{T}_k)$
14:    $V_k = Q_k R_k$
15: end while
16: Truncate $[\boldsymbol{\lambda}_k, V_k]$ to top $m$ eigenpairs
```

4. Application to Graph Data

In this section, we explain how to apply the Lanczos algorithm to graph data. Normal PCA creates principal components depending on all features, assuming that all features can be accessed in memory. For graph data, it is not possible to access all features at the same time. However, due to the structure of the feature space, namely, subgraph-supergraph relationships among patterns, we can solve the following weighted substructure mining problem [11]. Denote by $\boldsymbol{w} \in \Re^n$ example weights. The weighted mining enumerates the following pattern set,

$$P_{\boldsymbol{w}} = \{p \mid \left|\sum_{j=1}^{n} w_j x_{jp}\right| \geq \epsilon\}, \tag{5}$$

where ϵ is a predetermined constant. Therefore, we can quickly collect features highly correlated with a given vector $\boldsymbol{w}$. See next section for the mining algorithm. Our question here is how we can use this tool to compute principal components approximately without accessing all features.

To understand the problem better, let us first consider the PCA of d-dimensional vectorial data, i.e., $X \in \mathbb{R}^{n\times d}$. PCA is done by the eigendecomposition of the Gram matrix $A = XX^\top$. Let us consider the following intermediate variable

$$\boldsymbol{g}_k = X^\top \boldsymbol{q}_k.$$

where the i-th entry is described as

$$g_{ki} = \sum_{i=1}^{n} x_{ip} q_{kj}.$$

Substituting $A = XX^\top$ to the equations in Algorithm 2, it turns out that the Lanczos algorithm accesses the design matrix X through $\boldsymbol{g}_k$ and $X\boldsymbol{g}_k$. The reformulated pseudo code is summarized in Algorithm 3. What it implies is that the features corresponding to zero entries $\{i|g_{ki} = 0\}$ need not to be accessed in the k-th iteration. More aggressively, we can reduce the number of accessed features by introducing the tolerance threshold ϵ as

$$\{i \mid |g_{ki}| \geq \epsilon\}. \quad (6)$$

Alternatively, one can take the best ℓ features with largest $|g_{ki}|$. It provides a reasonable method to trade the number of accessed features and the accuracy of principal components.

Algorithm 3 PCA based on the Lanczos method

```
1:  Input: Design matrix $X \in \mathbb{R}^{n \times d}$
2:  Output: Principal components $Z \in \Re^{d \times m}$
3:  Initial: $\boldsymbol{r}_0 = \boldsymbol{q}_1;\ \beta_0 = 1;\ k = 0$
4:  while (4) is not true do
5:      $\boldsymbol{q}_k = \boldsymbol{r}_k / \beta_k$
6:      $k = k + 1$
7:      $\boldsymbol{g}_k = X^\top \boldsymbol{q}_k$                ▷ Intermediate variable
8:      $\alpha_k = \boldsymbol{g}_k^\top \boldsymbol{g}_k$
9:      $\boldsymbol{r}_k = X\boldsymbol{g}_k - \beta_{k-1}\boldsymbol{q}_{k-1} - \alpha_k \boldsymbol{q}_k$
10:     for $j = 1 : k - 1$ do
11:         $\boldsymbol{r}_k = \boldsymbol{r}_k - \boldsymbol{q}_j(\boldsymbol{q}_j^\top \boldsymbol{r}_k)$
12:     end for
13:     $\beta_k = \|\boldsymbol{r}_k\|$
14:     $[\boldsymbol{\lambda}_k, R_k] = EigenDecomposition(\boldsymbol{T}_k)$
15:     $V_k = Q_k R_k$
16: end while
17: Truncate $[\boldsymbol{\lambda}_k, V_k]$ to top $m$ eigenpairs
18: $Z = X^\top V_m$
```

For graph data, the features satisfying the criterion (6) can be enumerated by mining (5) with $\boldsymbol{w} = \boldsymbol{q}_k$. By incorporating the mining step to Algorithm 3, we finally arrive at our graph PCA algorithm (Algorithm 4). For indexing purposes, we collect mined patterns to the pattern pool P. Each principal components is indexed by the patterns correspond to its major elements. In analogy to text mining, each principal component corresponds to a "topic" and the associated patterns describe that topic.

Algorithm 4 Graph PCA

```
1:  Input: Graphs $G_1, \ldots, G_n$, Tolerance $\epsilon$
2:  Output: Patterns $P$, Principal components $Z \in \Re^{|P| \times m}$
3:  Initial: $\boldsymbol{q}_1 = \boldsymbol{1}/\sqrt{n};\ \boldsymbol{r}_0 = \boldsymbol{q}_1;\ \beta_0 = 1;\ k = 0;\ P = \emptyset$
4:  while (4) is not true do
5:      $\boldsymbol{q}_k = \boldsymbol{r}_k / \beta_k$
6:      $k = k + 1$
7:      $P_k = \{p \mid \left|\sum_{j=1}^{n} q_{kj} x_{jp}\right| \geq \epsilon\}$      ▷ Pattern search
8:      $P \leftarrow P \cup P_k$
9:      $X_P$: design matrix restricted to $P$
10:     $\alpha_k = \boldsymbol{q}_k^\top X_P X_P^\top \boldsymbol{q}_k$
11:     $\boldsymbol{r}_k = X_P X_P^\top \boldsymbol{q}_k - \beta_{k-1}\boldsymbol{q}_{k-1} - \alpha_k \boldsymbol{q}_k$
12:     for $j = 1 : k - 1$ do
13:         $\boldsymbol{r}_k = \boldsymbol{r}_k - \boldsymbol{q}_j(\boldsymbol{q}_j^\top \boldsymbol{r}_k)$
14:     end for
15:     $\beta_k = \|\boldsymbol{r}_k\|$
16:     $[\boldsymbol{\lambda}_k, R_k] = EigenDecomposition(\boldsymbol{T}_k)$
17:     $V_k = Q_k R_k$
18: end while
19: Truncate $[\boldsymbol{\lambda}_k, V_k]$ to top $m$ eigenpairs
20: $Z = X_P^\top V_m$
```

5 Optimal Pattern Search

In this section, we describe the weighted subgraph mining in detail. Our search strategy is a branch-and-bound algorithm that requires a canonical search space in which a whole set of patterns are enumerated without duplication. As the search space, we adopt the DFS code tree [20]. The basic idea of the DFS code tree is to organize patterns as a tree, where a child node has a supergraph of the pattern in its parent node. (Figure 2). A pattern is represented as a text string called the DFS (depth first search) code. The patterns are enumerated by generating the tree from the root to leaves using a recursive algorithm. To avoid duplications, node generation is systematically done by rightmost extensions. Algorithm 5 shows the pseudo code for the recursive algorithm.

For efficient search, it is important to minimize the size of the search space. To this aim, *tree pruning* is crucially important [9, 7]. As stated in (5), our task is to enumerate all the patterns whose gain function

$$s(p) = \left|\sum_{i=1}^{n} w_i x_{ip}\right|$$

is larger than ϵ. Suppose the search tree is generated up to the pattern p. If it is guaranteed that the score of any supergraph p' is not larger than ϵ, we can avoid the generation of downstream nodes without losing the optimal pattern. Our pruning condition is described as follows.

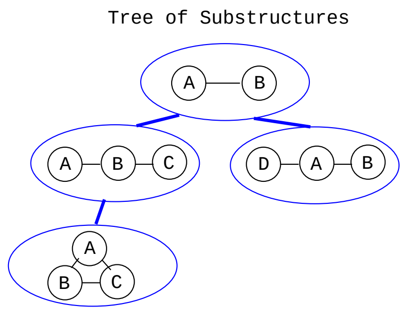

Figure 2. Schematic figure of the tree-shaped search space of graph patterns (i.e., the DFS code tree). To find the optimal pattern efficiently, the tree is systematically expanded by rightmost extensions.

Table 1. Summary of datasets.

	# data	# avg. atoms	# avg. bonds
EDKB-AR	146	19.5	21.1
EDKB-ER	131	19.2	20.7
EDKB-ES	59	18.2	19.7
CPDB	684	14.1	14.6
CAS	4337	29.9	30.9

Theorem 1 *Define*

$$s^+(p) = 2 \sum_{\{i|w_i \geq 0, p \subseteq G_i\}} |w_i| - \sum_{i=1}^{n} w_i \quad (7)$$

$$s^-(p) = 2 \sum_{\{i|w_i < 0, p \subseteq G_i\}} |w_i| + \sum_{i=1}^{n} w_i. \quad (8)$$

For any supergraph p' of p, the following bound holds.

$$s(p') \leq \max\{s^+(p), s^-(p)\} \quad (9)$$

Therefore, the search tree is pruned at p, if $\epsilon > \max\{s^+(p), s^-(p)\}$.

See [7] for the proof.

6 Experiments

In this section, we evaluate gPCA in terms of approximation error and computation time. Due to the truncation of the intermediate variables, the principal components obtained by gPCA are not equal to the true principal components which can only be obtained by total enumeration of patterns. Our goal is to build an approximated principal components to the true ones. We test our algorithm in three chemical datasets from the EDKB database are used[1]: EDKB-AR, ER and ES (Table 1). They stand for androgen receptors, estrogen receptors and E-assays, respectively. We employed relatively small datasets such that total enumeration is possible.

In this experiment, the number of principal components is set to three. In visualization and machine learning applications, it is desirable that the the similarity of graphs are appropriately represented by projected points. We first create "true" three-dimensional projections of graphs by total enumeration and PCA. Here, the minimum support is set to 2, and the pattern size is restricted up to 15 nodes. Denote by G_{true} the true Gram matrix, representing the dot product of three dimensional projections. It is compared with G, the Gram matrix of gPCA projections. Since they differ in scale, the Gram matrix is normalized as

$$\hat{G}_{ij} = \frac{G_{ij}}{\sqrt{G_{ii}G_{jj}}}.$$

Then, the approximation error is measured using the Frobenius norm as $\|\hat{G} - \hat{G}_{true}\|$. The smaller the approximation error means the reconstruction of the subspace closer to the true Gram matrix. In approximating the intermediate variable (6), we employed the top-L approximation rather than fixing the threshold ϵ.

Figures 4, 5 and 6 show the approximation error of gPCA in the EDKB-ES, EDKB-ER and EDKB-AR dataset, respectively. As L increases, the Gram matrix is more accu-

Algorithm 5 Pattern search algorithm

```
1:  procedure PATTERN SEARCH
2:      $P \leftarrow \emptyset$
3:      for $p \in$ DFS codes with single nodes do
4:          project($p$)
5:      end for
6:      return $\mathcal{P}$
7:  end procedure
8:  function PROJECT($p$)
9:      if $p$ is not a minimum DFS code then
10:         return
11:     end if
12:     if pruning condition (9) holds then
13:         return
14:     end if
15:     if $p$ satisfies the condition (6) then
16:         $P \leftarrow P \cup \{p\}$
17:     end if
18:     for $p' \in$ rightmost extensions of $p$ do
19:         project($p'$)
20:     end for
21: end function
```

[1]http://edkb.fda.gof/databasedoor.html

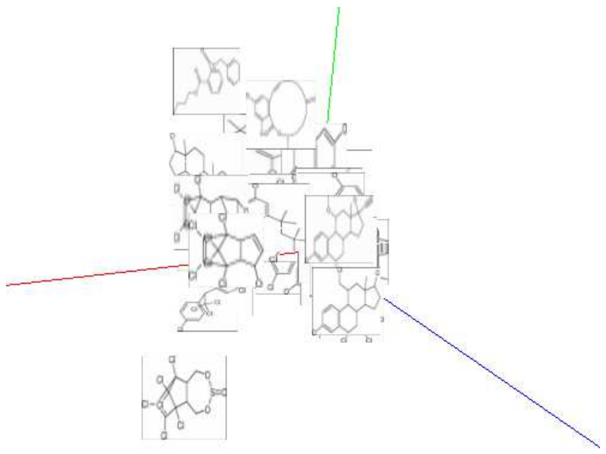

Figure 3. A three-dimensional plot of projected points in the EDKB-ER dataset.

rately approximated. The projected points are illustrated in Figure 3. Also, the patterns associated with principal components are shown in Figures 7, 8 and 9. It is observed that similar subgraph patterns are clustered together in the same column, but the patterns in different columns are distinctly different. Frequent patterns are often small and boring. When the patterns are listed according to frequency, one has to go down the list up to the bottom to find meaningful features. Our patterns are not immediately interpretable like words in text mining. Nevertheless our PCA-based patterns look much more meaningful because they are relatively large and capture complex features.

The computational time for larger datasets are measured and shown in Table 2. For such medium-sized datasets, the computational time stays within the tractable level (e.g., several hours) in an ordinary PC. It is possible to reduce the computational time by maintaining the whole search tree in each iteration [17]. However, it would take much more memory than the current implementation.

7 Conclusion

In this paper, we presented an iterative mining method that can capture the patterns corresponding to major entries of principal components. An advantage of iterative mining methods is that the database is processed with different criteria, leading to several different pattern sets. Compared to frequent mining with only one criterion, patterns by iterative mining can characterize different aspects of the graph database. In supervised learning task, there have been proposed several iterative mining algorithms [11, 14]. There are several unsupervised clustering algorithms [19, 18] as

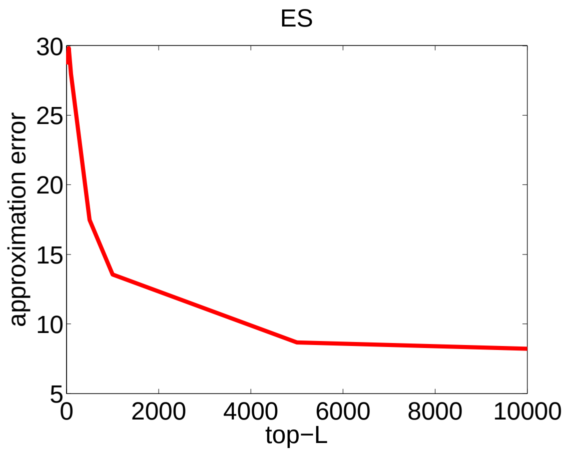

Figure 4. Approximation error using top 3 principal components in the EDKB-ES dataset.

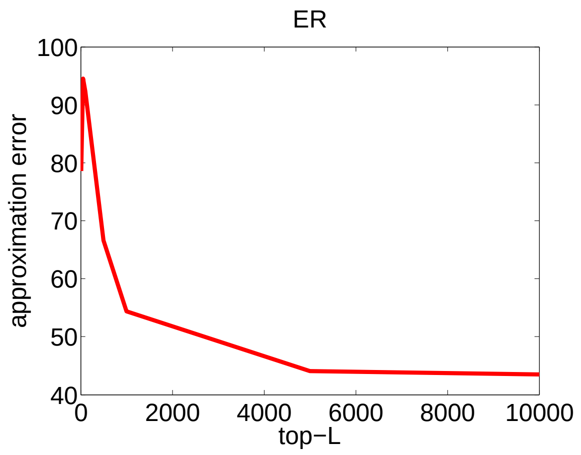

Figure 5. Approximation error using top 3 principal components in the EDKB-ER dataset.

Table 2. Time for finding top 3 principal components in the CAS and CPDB dataset.

	CAS		CPDB	
L	time (sec)	# iter	time (sec)	# iter
10	9.038	11	1.611	4
50	46.55	6	6.472	4
100	110.9	6	12.21	4
500	702.8	6	62.57	4
1000	2227	4	90.79	4

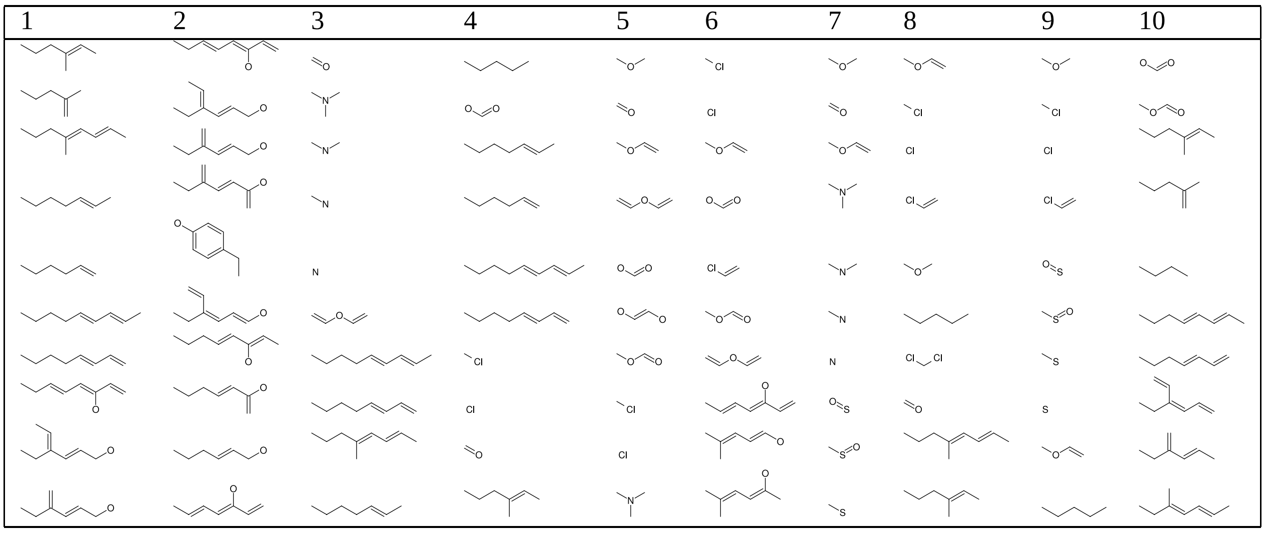

Figure 7. Patterns associated with the top 10 principal components in the EDKB-ER dataset.

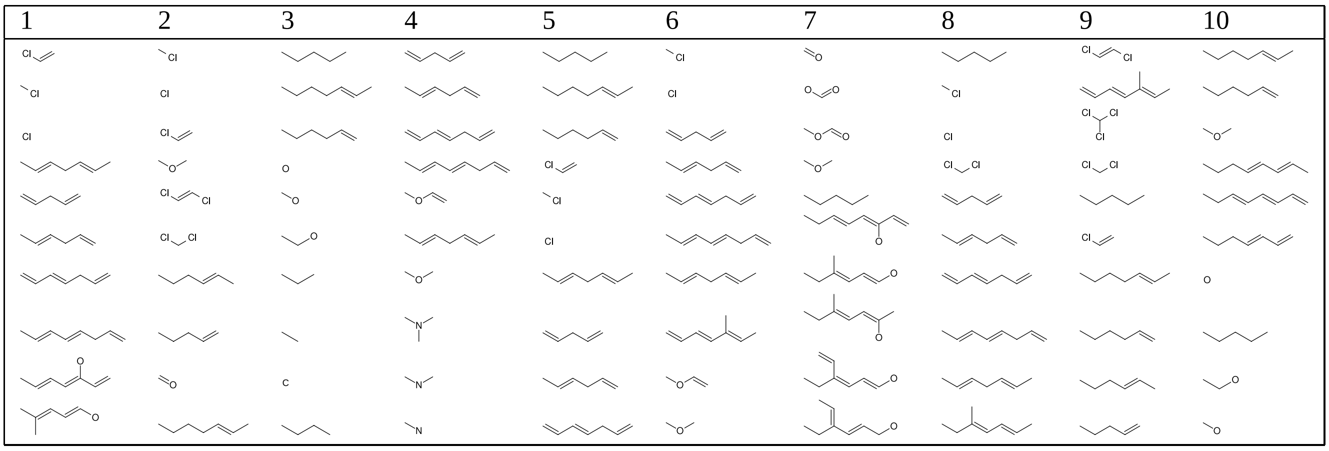

Figure 8. Patterns associated with the top 10 principal components in the EDKB-ES dataset.

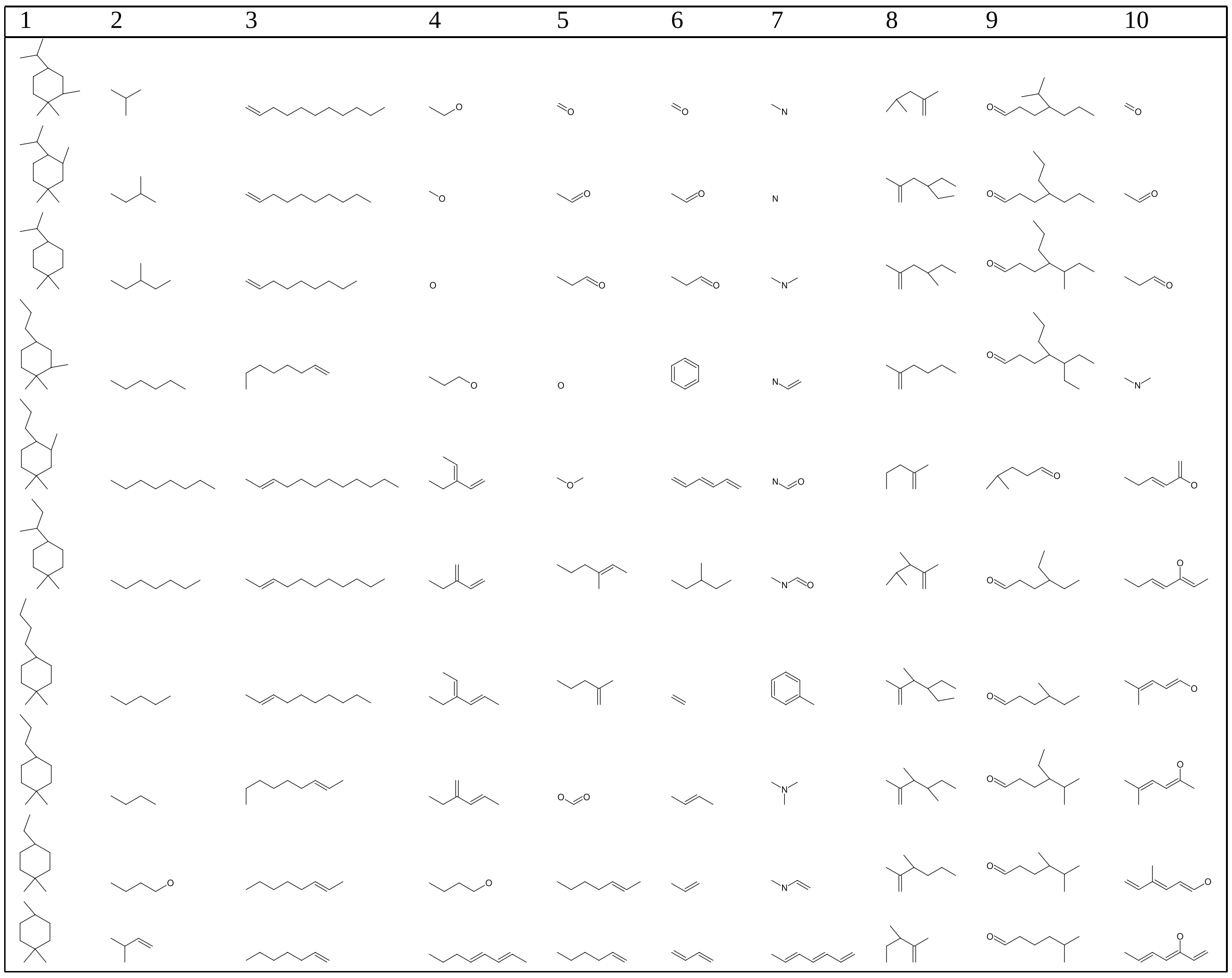

Figure 9. Patterns associated with the top 10 principal components in the EDKB-AR dataset.

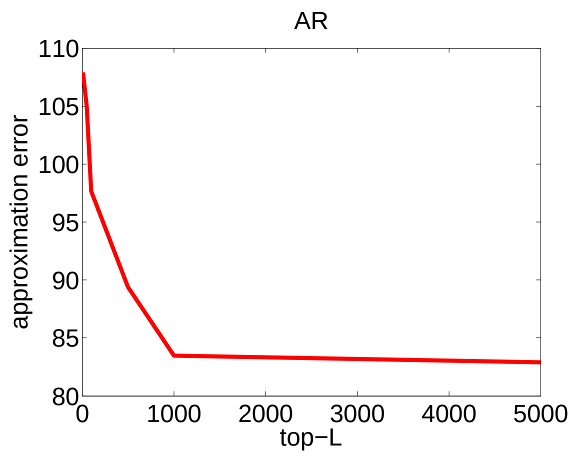

Figure 6. Approximation error using top 3 principal components in the EDKB-AR dataset.

well, but they are essentially similar to the supervised methods in that they use putative class labels assigned by the EM algorithm. We pursued a different path, namely the coupling of the Lanczos method and the weighted subgraph mining, which does not rely on putative class labels.

Since PCA is the most fundamental dimensionality reduction algorithm, gPCA can serve as a basic template for creating mining algorithms for related problems, such as canonical component analysis, projection pursuit and non-negative matrix decomposition. Probablistic variants such as probabilistic latent semantic analysis are also in sight. Our mining algorithm can be combined with different kind of mining algorithms such as itemset mining, sequence mining and tree mining.

References

[1] M. W. Berry. Large-scale sparse singular value computations. *The International Journal of Supercomputer Applications*, 6(1):13–49, Spring 1992.

[2] M. W. Berry, S. T. Dumais, and G. W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37:573–595, 1995.

[3] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41, 1990.

[4] G. H. Golub and C. F. V. Loan. *Matrix computations*. Johns Hopkins University Press, 1996.

[5] A. Inokuchi. Mining generalized substructures from a set of labeled graphs. In *Proceedings of the 4th IEEE Internatinal Conference on Data Mining*, pages 415–418. IEEE Computer Society, 2005.

[6] I. C. F. Ipsen and C. D. Meyer. The idea behind Krylov methods. *American Mathematical Monthly*, 105(10):889–899, 1998.

[7] T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Advances in Neural Information Processing Systems 17*, pages 729–736. MIT Press, 2005.

[8] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand*, 45:255–282, 1950.

[9] S. Morishita. Computing optimal hypotheses efficiently for boosting. In *Discovery Science*, pages 471–481. Springer, 2001.

[10] S. Nijssen and J. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 647–652. ACM Press, 2004.

[11] S. Nowozin, K. Tsuda, T. Uno, T. Kudo, and G. Bakir. Weighted substructure mining for image analysis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2007.

[12] Y. Saad. On the rates of convergence of the Lanczos and the block Lanczos methods. *SIAM J. Num. Anal.*, 17:687–706, 1980.

[13] H. Saigo, T. Kadowaki, and K. Tsuda. A linear programming approach for molecular QSAR analysis. In *International Workshop on Mining and Learning with Graphs (MLG)*, pages 85–96, 2006.

[14] H. Saigo, N. Krämer, and K. Tsuda. Partial least squares regression for graph mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2008.

[15] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[16] E. Sjöström. *Singular value computations for Toeplitz matrices*. PhD thesis, Linköping University, 1996.

[17] K. Tsuda. Entire regularization paths for graph data. In *Proceedings of the 24th International Conference on Machine Learning*, pages 919–926, 2007.

[18] K. Tsuda and T. Kudo. Clustering graphs by weighted substructure mining. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 953–960. ACM Press, 2006.

[19] K. Tsuda and K. Kurihara. Graph mining with variational dirichlet process mixture models. In *SIAM Conference on Data Mining (SDM)*, 2008.

[20] X. Yan and J. Han. gSpan: graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, pages 721–724. IEEE Computer Society, 2002.

Appendix: Eigendecomposition using the Lanczos algorithm

In this section we briefly review how to solve an eigendecomposition problem. We hilight the Lanczos algorithm as an efficient subroutine for solving this problem.

For simplicity, we consider the eigendecomposition of a Gram matrix $\boldsymbol{A}$ into a diagonal matrix $\boldsymbol{\lambda}$ and an orthogonal basis $\boldsymbol{Q}$ such that the following relationship holds;

$$\boldsymbol{AV} = \boldsymbol{V\lambda}, \tag{10}$$

where $\boldsymbol{\lambda} = diag(\lambda_1, \lambda_1, \ldots, \lambda_n)$. Since $\boldsymbol{A}$ is symmetric, it can be tridiagonalized such that the following relationship holds

$$\boldsymbol{AQ} = \boldsymbol{QT}, \tag{11}$$

where $\boldsymbol{Q}$ is an orthonormal basis. Moreover, tridiagonal matrix $\boldsymbol{T}$ can be decomposed into the Schur form

$$\boldsymbol{TR} = \boldsymbol{RE}, \tag{12}$$

where $\boldsymbol{R}$ is an orthonormal basis and $\boldsymbol{E}$ is a diagonal matrix. Using equations (11) and (12), $\boldsymbol{A}$ can be rewritten as

$$\boldsymbol{A} = \boldsymbol{QTQ}^{-1} = (\boldsymbol{QR})\,\boldsymbol{E}\,(\boldsymbol{QR})^{-1} = \boldsymbol{ZEZ}^{-1} \tag{13}$$

This is an eigendecomposition of $\boldsymbol{A}$. Since eigendecomposition of a symmetric matrix is uniquely determined, diagonal matrix $\boldsymbol{E}$ and orthogonal basis $\boldsymbol{Z}$ coincides with $\boldsymbol{\lambda}$ and $\boldsymbol{V}$ in (10), respectively. Therefore, it is clear that we can split the eigendecomposition into two steps; i) tridiagonalization and ii) Schur decomposition. For the latter part, we can employ Givens rotations to perform Schur decomposition on a tridiagonal matrix in linear time. Now the problem is how to efficiently compute the former part.

The following Lanczos algorithm generates a sequence of tridiagonal matrices $\boldsymbol{T}_k$ from $\boldsymbol{A}$ with the property that eigenvalues of $\boldsymbol{T}_k$ are progressively better estimates of the extreme eigenvalues of $\boldsymbol{A}$. The tridiagonal matrix generated by the Lanczos algorithm is written as

$$\boldsymbol{T}_k = \begin{bmatrix} \alpha_1 & \beta_1 & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \beta_{k-1} \\ 0 & \ldots & \beta_{k-1} & \alpha_k \end{bmatrix}$$

By equating columns in $\boldsymbol{AQ} = \boldsymbol{QT}$, we find

$$\boldsymbol{Aq}_k = \beta_{k-1}\boldsymbol{q}_{k-1} + \alpha_k \boldsymbol{q}_k + \beta_k \boldsymbol{q}_{k+1}, \quad \beta_0 \boldsymbol{q}_0 \equiv 0.$$

From the orthonormality condition of $\boldsymbol{q}$, we immediately get

$$\alpha_k = \boldsymbol{q}_k^\top \boldsymbol{Aq}_k.$$

Moreover,

$$\beta_k = \frac{\boldsymbol{Aq}_k - \beta_{k-1}\boldsymbol{q}_{k-1} - \alpha_k \boldsymbol{q}_k}{\boldsymbol{q}_{k+1}},$$

as long as β's numerator is nonzero. If the numerator were zero, then we have successfully obtained the invariant subspace. Notice that α_k and β_k can be efficiently calculated one another with the update of $\boldsymbol{q}_k$. This whole procedure is described in Algorithm 1. The q_k are called as *Lanczos vectors*, and stored for the later use.

As can be seen from the pseudocode, the Lanczos method can directly calculate the tridiagonal matrix $\boldsymbol{T} = \boldsymbol{Q}^\top \boldsymbol{AQ}$. Moreover, we neither need to modify $\boldsymbol{A}$, nor even need a direct access to $\boldsymbol{A}$ provided that matrix-vector multiplication $\boldsymbol{Aq}$ is available. Note that Householder tridiagonalization or Givens rotation can also tridiagonalize $\boldsymbol{A}$. However, these methods need to update $\boldsymbol{A}$, and therefore not being applicable when $\boldsymbol{A}$ is large and sparse.