

Graph Mining with Variational Dirichlet Process Mixture Models

Koji Tsuda*

Kenichi Kurihara[†]

Abstract

Graph data such as chemical compounds and XML documents are getting more common in many application domains. A main difficulty of graph data processing lies in the intrinsic high dimensionality of graphs, namely, when a graph is represented as a binary feature vector of indicators of all possible subgraph patterns, the dimensionality gets too large for usual statistical methods. We propose a nonparametric Bayesian method for clustering graphs and selecting salient patterns at the same time. Variational inference is adopted here, because sampling is not applicable due to extremely high dimensionality. The feature set minimizing the free energy is efficiently collected with the DFS code tree, where the generation of useless subgraphs is suppressed by a tree pruning condition. In experiments, our method is compared with a simpler approach based on frequent subgraph mining, and graph kernels.

1 Introduction

Graphs are general and powerful data structures used to represent diverse kinds of objects. In many cases, real-world data are 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. In learning from graph data, one can rely on the similarity measures derived from graph alignment [9] or graph kernels [5]. However, one drawback is that the features used in learning are implicitly defined, and derived clusters are hard to interpret. Another approach is based on *graph mining*, where a set of small graphs (i.e., patterns) is used to represent a graph. Specifically, each graph is represented as a binary vector of pattern indicators (Figure 1). Graph mining is especially popular in chemoinformatics, where the task is to classify chemical compounds [6, 4]. When all possible subgraphs are used, the dimensionality of the feature space is too large for usual statistical methods. Therefore, *feature selection* is a central issue in graph mining algorithms [12, 2].

So far, probabilistic inference has rarely been applied to graph data (see e.g. [14]). One reason would be that the main interest of researchers has been mainly in sophisticated and complex algorithms for pattern enumeration. However, a set of graphs belonging to the same semantic category (e.g., chemical compounds that inhibit estrogen receptors) have great diversity in size and edge connections. To capture such diversity properly and deal with uncertainty in inference, probabilistic methods have to be introduced in this field. As a pioneering attempt, Tsuda and Kudo proposed an EM-based method for clustering graphs, where a set of salient patterns are efficiently collected based on latent cluster labels [12]. However, the number of clusters has to be specified a priori, and the model selection procedure is not discussed in their paper.

Dirichlet process (DP) mixture models have attracted much attention recently, because it is applicable even if the number of clusters is not known. This paper addresses the problem of learning a DP mixture model in the high dimensional feature space of graph data. In particular, we focus on variational inference [1, 8] due to its efficiency. To limit the dimensionality, it is necessary to formulate a feature saliency criterion and develop a search algorithm to find best patterns. In the context of DP mixtures, feature selection has not been an active area of research so far. One notable exception is by Kim et al. [7], but their method is based on sampling and not amenable to graph data, because one has to sample from the whole feature set. We propose a simpler approach that selects features by minimizing the variational free energy.

To find the best patterns quickly, we need a canonical search space in which a whole set of patterns are enumerated without duplication. To this aim, we adopt the DFS code tree [15], a standard method developed for a popular frequent subgraph mining algorithm, gspan. To keep the search space small, an effective tree pruning condition is crucial. We design a new criterion specifically for the variational free energy.

In the rest of this paper, we first introduce the DP mixture model based on the stick-breaking representation (Section 2). Then, our feature selection algorithm is introduced (Section 3), and implemented on the DFS code tree (Section 4). In experiments (Section 5), we

*Max Planck Institute for Biological Cybernetics, Tübingen, Germany

[†]Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan

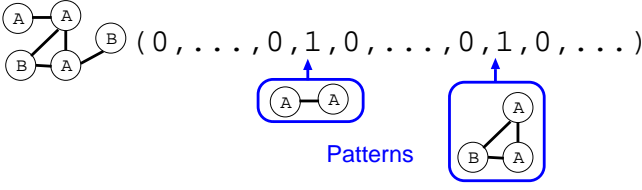


Figure 1: Feature space based on subgraph patterns. The feature vector consists of binary pattern indicators.

compare our method with a conventional method that collects frequent patterns first and learns a DP mixture afterwards. It will be shown that our method can produce better clusters with fewer features.

2 Dirichlet Process Binomial Mixture Model

In this section, a variational algorithm to learn a DP mixture of binomial distributions is briefly reviewed. We basically replace Gaussian distributions in [8] with binomials. Our data is represented as d -dimensional binary vectors $X = \{\mathbf{x}_i\}_{i=1}^n$, $\mathbf{x}_i \in \{0, 1\}^d$. The binomial mixture model is described as $p(\mathbf{x}_i|\Theta, \pi) = \sum_{l=1}^c p(\mathbf{x}_i, y_i = l|\Theta, \pi)$, where each component is a multivariate binomial distribution,

$$(2.1) \quad p(\mathbf{x}_i, y_i = l|\Theta, \pi) = \pi_l \prod_{k=1}^d \theta_{lk}^{x_{ik}} (1 - \theta_{lk})^{1-x_{ik}},$$

where π_l is the mixture weight of component l , c is the number of components and $y_i \in \{1, \dots, c\}$ is a latent variable. A Dirichlet process mixture model in the stick-breaking representation can be viewed as possessing an infinite number of components with random mixing weights [11]. Denote by $v_l \in [0, 1]$ the l -th stick length. The stick-breaking representation is represented as

$$(2.2) \quad \pi_l = v_l \prod_{j=1}^{l-1} (1 - v_j),$$

$$(2.3) \quad p_{v_l}(v_l) = \text{Beta}(v_l; 1, \alpha),$$

$$(2.4) \quad p_{\theta_{lk}}(\theta_{lk}) = \text{Beta}(\theta_{lk}; a, b).$$

Our task is to infer the posterior distribution of assignments, $p(y_i|X)$. Since $p(y_i|X)$ cannot be solved analytically, a typical method to infer the posterior is the Gibbs sampler. However, it is too inefficient for graph clustering. Thus, we take the variational approach [1]. For variational inference, the intractable posterior is approximated by factorized variational distributions,

$$(2.5) \quad q(\mathbf{y}, \mathbf{v}, \Theta) = \left[\prod_{i=1}^n q_{y_i}(y_i) \right] \left[\prod_{l=1}^L q_{v_l}(v_l) \prod_{k=1}^d q_{\theta_{lk}}(\theta_{lk}) \right],$$

where L is a truncation level of the Dirichlet process mixture. Hence, we infer $q_{y_i}(y_i)$ instead of $p(y_i|X)$.

The variational posterior is optimized by minimizing the KL divergence $D[q(\mathbf{y}, \mathbf{v}, \Theta), p(\mathbf{y}, \mathbf{v}, \Theta|X)]$, or equivalently minimizing the free energy,

$$\mathcal{F}_L \equiv \mathbb{E}_q[\log q(\mathbf{y}, \mathbf{v}, \Theta)] - \mathbb{E}_q[\log p(\mathbf{y}, \mathbf{v}, \Theta, X)],$$

where \mathbb{E}_q denotes the expectation with respect to $q(\mathbf{y}, \mathbf{v}, \Theta)$. \mathcal{F}_L is the free energy of an L -truncated Dirichlet process. Although the free energy of an infinite Dirichlet process is defined as limit $\mathcal{F} = \lim_{L \rightarrow \infty} \mathcal{F}_L$, it is intractable due to infinite sum. In a similar fashion to [8], we introduce a new constant T , and constrain $q_{y_i}(y_i = l) = 0$ for $l > T$. This constraint makes \mathcal{F} tractable. Furthermore, \mathcal{F} is nested over T as [8]. Hence, our free energy is given as,

$$(2.6) \quad \mathcal{F} = \sum_{l=1}^T \left\{ D[q_{v_l}(v_l), p_{v_l}(v_l)] + \sum_{k=1}^d D[q_{\theta_{lk}}(\theta_{lk}), p_{\theta_{lk}}(\theta_{lk})] \right\} + \sum_{i=1}^n \mathbb{E}_q[\log q_{y_i}(y_i)] - \mathbb{E}_q[\log p(\mathbf{x}_i, y_i|\Theta, \mathbf{v})].$$

We show the details of derivation in Appendix A. Although the constraint we imposed is slightly different from one given in [8], our free energy leads to simpler update rules. Furthermore, the feature selection criterion (3.20) is greatly simplified. According to our pilot experiments, the difference of constraints did not produce any meaningful difference in experimental results.

From (2.6), q that minimizes \mathcal{F} is obtained as,

$$(2.7) \quad q_{v_l}(v_l) = \begin{cases} \text{Beta}(v_l; \gamma_{l1}, \gamma_{l2}) & (l \leq T) \\ p_{v_l}(v_l) & (l > T), \end{cases}$$

$$(2.8) \quad q_{\theta_{lk}}(\theta_{lk}) = \begin{cases} \text{Beta}(\theta_{lk}; a_{lk}, b_{lk}) & (l \leq T) \\ p_{\theta_{lk}}(\theta_{lk}) & (l > T), \end{cases}$$

where $\gamma_{l1} = 1 + \sum_{i=1}^n q_{y_i}(y_i = l)$, $\gamma_{l2} = \alpha + \sum_{i=1}^n \sum_{j=l+1}^T q_{y_i}(y_i = j)$, and

$$(2.9) \quad a_{lk} = a + \sum_{i=1}^n q_{y_i}(y_i = l) x_{ik},$$

$$(2.10) \quad b_{lk} = b + \sum_{i=1}^n q_{y_i}(y_i = l) (1 - x_{ik}).$$

Based on these solutions, the optimal cluster assignment is obtained as $q_{y_i}(y_i = l) \propto \exp(s_{il})$ where Ψ is the di-

gamma function and

$$(2.11) \quad s_{il} = \sum_{k=1}^d \{x_{ik}\Psi(a_{lk}) + (1-x_{ik})\Psi(b_{lk}) - \Psi(a_{lk} + b_{lk})\} \\ + \Psi(\gamma_{l1}) - \Psi(\gamma_{l1} + \gamma_{l2}) + \sum_{j=1}^{l-1} \{\Psi(\gamma_{j2}) - \Psi(\gamma_{j1} + \gamma_{j2})\}.$$

To minimize the free energy, q_y and $\{q_\theta, q_v\}$ are alternately updated.

3 Feature Selection

In substructure representation (Figure 1), the number of dimensionality d is intractably large. In this section, we propose a feature selection method to obtain a reduced feature set A of prespecified size $|A| = m$. In Bayesian inference, the new variable A can be inferred either as a latent variable [7] or as a hyperparameter [13]. As an example of the former approach, Kim et al. [7] proposed to introduce new Bernoulli latent variables to indicate selected features, and the posterior distribution is inferred by Gibbs sampling. However, this approach is not amenable to our problems, because one has to sample from the whole feature set. In the following, we regard A as a hyperparameter, and estimate it by minimizing the free energy.

Our algorithm incorporated with a feature selection updates q and A alternately to minimize the free energy. We first derive q with fixed A . We also propose a feature selection with fixed q later.

3.1 Posteriors with Selected Features Before we go to the detail of feature selection, we modify the binomial cluster model (2.1), and derive posteriors q_y , q_θ and q_v with fixed A . We also show that given A of size m the computational complexity of updating q is reduced to $O(m)$ from $O(d)$ of model (2.1) where d is the number of all features. The reduction of complexity is not trivial because we need to keep all of d features in our model so that we can select reduced feature set A from d features by minimizing the free energy.

The binomial cluster model (2.1) is modified as follows. For $k \in A$, we keep $q_{\theta_{lk}}(\theta_{lk})$ for each l , but for $k \notin A$, a baseline distribution $q_{\theta_{0k}}(\theta_{0k})$ is used for all $l = 1, \dots, T$. The prior distribution is defined as $p_{\theta_{0k}}(\theta_{0k}) = \text{Beta}(\theta_{0k}; a, b)$. The joint model of the feature vector and the class label is described as

$$(3.12) \quad p_A(\mathbf{x}_i, y_i = l | \theta, v) = \pi_l \prod_{k \in A} \theta_{lk}^{x_{ik}} (1 - \theta_{lk})^{1-x_{ik}} \\ \prod_{k \notin A} \theta_{0k}^{x_{ik}} (1 - \theta_{0k})^{1-x_{ik}}.$$

Adding new variables, $\{\theta_{0k}\}_{k=1}^d$, the variational distribution becomes

$$(3.13) \quad q(\mathbf{y}, \mathbf{v}, \Theta, \Theta_0) = \left[\prod_{i=1}^n q_{y_i}(y_i) \right] \left[\prod_{l=1}^L q(v_l) \right] \\ \left[\prod_{k=1}^d \prod_{l=1}^L q(\theta_{lk}) \right] \left[\prod_{k=1}^d q(\theta_{0k}) \right].$$

The free energy becomes

$$(3.14) \quad \mathcal{F} = \sum_{l=1}^T \left\{ D[q(v_l), p(v_l)] + \sum_{k=1}^d D[q(\theta_{lk}), p(\theta_{lk})] \right\} \\ + \sum_{k=1}^d D[q(\theta_{0k}), p(\theta_{0k})] \\ + \sum_{i=1}^n \mathbb{E}_q[\log q_{y_i}(y_i)] - \mathbb{E}_q[\log p_A(\mathbf{x}_i, y_i | \Theta, \mathbf{v})].$$

We need to minimize \mathcal{F} jointly over q_v, q_θ, q_y and A . Among them, q_v is solved as in the previous section, and q_θ is analytically solved as

$$(3.15) \quad q_{\theta_{lk}}(\theta_{lk}) = \begin{cases} \text{Beta}(\theta_{lk}; a_{lk}, b_{lk}) & (k \in A) \\ \text{Beta}(\theta_{lk}; a, b) & (k \notin A), \end{cases}$$

$$(3.16) \quad q_{\theta_{0k}}(\theta_{0k}) = \begin{cases} \text{Beta}(\theta_{0k}; a_{0k}, b_{0k}) & (k \notin A) \\ \text{Beta}(\theta_{0k}; a, b) & (k \in A). \end{cases}$$

where a_{lk}, b_{lk} are defined as (2.9) and (2.10) and $a_{0k} = a + \sum_{i=1}^n x_{ik}$, $b_{0k} = b + \sum_{i=1}^n (1 - x_{ik})$.

Plugging the solutions into \mathcal{F} , the remaining variables are q_y and A , which are updated alternately. We optimize $q_{y_i}(y_i)$ fixing A . The solution is obtained as $q_{y_i}(y_i) \propto \exp(s_{il})$ where

$$(3.17) \quad s_{il} = \sum_{k \in A} \{x_{ik}\Psi(a_{lk}) + (1-x_{ik})\Psi(b_{lk}) - \Psi(a_{lk} + b_{lk})\} \\ + \sum_{k \notin A} \{x_{ik}\Psi(a_{0k}) + (1-x_{ik})\Psi(b_{0k}) - \Psi(a_{0k} + b_{0k})\} \\ + \Psi(\gamma_{l1}) - \Psi(\gamma_{l1} + \gamma_{l2}) + \sum_{j=1}^{l-1} \{\Psi(\gamma_{j2}) - \Psi(\gamma_{j1} + \gamma_{j2})\}.$$

Since we can omit terms not depending on l in s_{il} , we can simplify it as,

$$(3.18) \quad s_{il} = \sum_{k \in A} \{x_{ik}\Psi(a_{lk}) + (1-x_{ik})\Psi(b_{lk}) - \Psi(a_{lk} + b_{lk})\} \\ + \Psi(\gamma_{l1}) - \Psi(\gamma_{l1} + \gamma_{l2}) + \sum_{j=1}^{l-1} \{\Psi(\gamma_{j2}) - \Psi(\gamma_{j1} + \gamma_{j2})\}.$$

Algorithm 1 Variational learning of a Dirichlet process mixture model with feature selection

- 1: Initialize $q_{y_i}(y_i)$ randomly.
 - 2: **repeat**
 - 3: Update $a_{lk}, b_{lk}, \gamma_{l1}$ and γ_{l2} for $k \in A, l = 1, \dots, T$.
 - 4: Update A by selecting the top m features with the largest gain (3.21) ▷ Pattern Search
 - 5: Cluster assignment $q_{y_i}(y_i)$ is updated as (3.18)
 - 6: **until** the free energy \mathcal{F} converges
-

Importantly, the cluster assignment q_y is computed solely from features in A . Hence, to update q_y , we only need to update $a_{lk}, b_{lk}, \gamma_{l1}$ and γ_{l2} for $l = 1 \dots T$ and $k \in A$. The computational complexity of updating q_y in terms of features is $O(m)$ although that without feature selection is $O(d)$.

3.2 Criterion for Feature Selection The feature set A is optimized fixing q_y . Ignoring the terms not depending on A , the free energy is described as

$$(3.19) \quad \mathcal{F}' = \sum_{k \in A} \sum_{l=1}^T D[q_{\theta_{lk}}(\theta_{lk}), p_{\theta_{lk}}(\theta_{lk})] + \sum_{k \notin A} D[q_{\theta_{0k}}(\theta_{0k}), p_{\theta_{0k}}(\theta_{0k})] - \sum_i \mathbb{E}_q[\log p_A(\mathbf{x}_i, y_i | \Theta, \mathbf{v})].$$

Substituting the optimal q_θ and q_v , it further simplifies as

$$(3.20) \quad \mathcal{F}' = - \sum_{k \in A} \sum_{l=1}^T \log B(a_{lk}, b_{lk}) - \sum_{k \notin A} \log B(a_{0k}, b_{0k}) + \text{const},$$

where B is the beta function. For the detail from (3.19) to (3.20), see Appendix B. To minimize \mathcal{F} under the constraint $|A| = m$, it is sufficient to sort the features according to the gain function

$$(3.21) \quad g_k = \sum_{l=1}^T \log B(a_{lk}, b_{lk}) - \log B(a_{0k}, b_{0k})$$

in descending order and take the first m features. The algorithm described in this section is summarized in Algorithm 1.

4 Finding Optimal Patterns

To apply our algorithm to graph data, we have to find the best m patterns that maximize the gain function

(3.21). First, let us define some notations. Given a graph database $\mathcal{G} = \{G_i\}_{i=1}^n$, let \mathcal{K} denote the set of all patterns, i.e., the set of all subgraphs included in at least one graph in \mathcal{G} . Then, each graph G_i is encoded as a feature vector $\mathbf{x}_i = (x_{ik})_{k \in \mathcal{K}}, x_{ik} = I(k \subseteq G_i)$, where $k \subseteq G_i$ denotes that k is a subgraph of G_i , and $I(\cdot)$ is 1 if the condition inside is true and 0 otherwise.

Our search strategy 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 [15]. The basic idea of the DFS code tree is to organize patterns as a tree, where a child node has a supergraph of the parent's pattern (Figure 2, left). 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. See Appendix C for details.

For efficient search, it is important to minimize the size of the search space. To this aim, *tree pruning* is crucial: Suppose the search tree is generated up to the pattern k and denote by g_k^* the gain of the m -th best pattern among the ones observed so far. If it is guaranteed that $g_{k'}$ of any supergraph k' is not larger than g_k^* , we can avoid the generation of downstream nodes without losing the m best patterns.

Let us define $\tilde{a}_{lk} = \sum_{i=1}^n q_{y_i}(y_i = l) x_{ik}$ and $\tilde{a}_{0k} = \sum_{i=1}^n x_{ik}$. Using a new function $\varphi(x, z) = \log B(a + x, b + z - x)$, the gain function is simply described as

$$g_k = \sum_{l=1}^T \varphi(\tilde{a}_{lk}, \beta_l) - \varphi(\tilde{a}_{0k}, n),$$

where $\beta_l = \sum_{i=1}^n q_{y_i}(y_i = l)$. Also define $a^* := \text{argmin}_a \varphi(a, n)$. Now, we propose the following pruning condition.

THEOREM 4.1. *The inequality $g_{k'} < g_k^*$ holds for any supergraph k' of k , if*

$$g_k^* > \max\left\{ \sum_{l=1}^T \varphi(0, \beta_l), \sum_{l=1}^T \varphi(\tilde{a}_{lk}, \beta_l) \right\} - \mu(\tilde{a}_{0k}),$$

where

$$\mu(\tilde{a}_{0k}) = \begin{cases} \varphi(a^*, n), & (\tilde{a}_{0k} \geq a^*) \\ \varphi(\tilde{a}_{0k}, n), & (\tilde{a}_{0k} < a^*). \end{cases}$$

Proof. For any supergraph $k' \supseteq k$, $\tilde{a}_{lk'} \leq \tilde{a}_{lk}$ and $\tilde{a}_{0k'} \leq \tilde{a}_{0k}$. Since $\varphi(x, z)$ is a convex function of x as shown in Figure 2, right, the first term of $g_{k'}$ is bounded from above as

$$\sum_{l=1}^T \varphi(\tilde{a}_{lk'}, \beta_l) \leq \max\left\{ \sum_{l=1}^T \varphi(0, \beta_l), \sum_{l=1}^T \varphi(\tilde{a}_{lk}, \beta_l) \right\}.$$

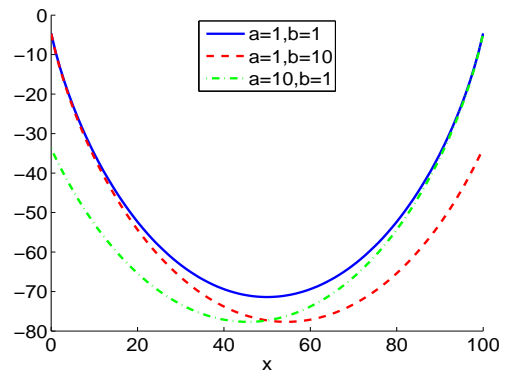
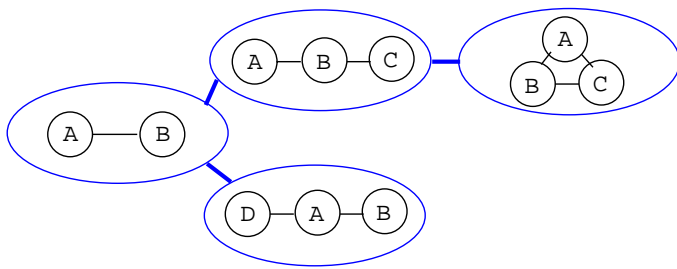


Figure 2: (Left) Schematic figure of the tree-shaped search space of graph patterns (i.e., the DFS code tree). (Right) Plot of $\varphi(x, z)$, $z = 100$.

The second term is bounded from below as $\varphi(\tilde{a}_{0k'}, n) \geq \mu(\tilde{a}_{0k})$. Then, the gain of any supergraph is bounded as $g_{k'} \leq \max\{\sum_{l=1}^T \varphi(0, \beta_l), \sum_{l=1}^T \varphi(\tilde{a}_{lk}, \beta_l)\} - \mu(\tilde{a}_{0k})$.

5 Experiments

In this section, our method is evaluated in terms of classification accuracy and generalization performance. In the former case, we evaluate our method as a clustering method. We prepare a dataset with known class labels, and compare our clustering result with the ground-truth label. In the latter case, we treat our method as a density estimator on graph data. A graph dataset is divided into training and test sets. Our DP mixture model is learned from the training set, and the likelihood of test graphs is measured.

In both cases, our method is compared with a *baseline method* that constructs the feature space explicitly with m most frequent patterns and learns a DP mixture afterwards (i.e., feature selection by frequency). It is clearly suboptimal because it cannot take latent clusters into account. However, when one would like to apply a DP mixture to graphs, this method would be the first choice to try, because frequent substructure mining algorithms such as *gspan* [15] are widely available. In fact, feature selection by frequency is adopted in [6, 4], and even claimed to be effective for classification in [3].

5.1 Classification Accuracy To evaluate classification accuracy, our clusters need to be matched with true categories. We used the RNA graph dataset (Figure 3) consisting of 80 (30+50) labeled graphs of the two families, Intron GP I and RNase bact a [12]. An RNA is a single-stranded chain of four kinds of nucleotides (A,C,G,U), which takes a complicated shape via hybridization of A-U and C-G pairs (optionally G-

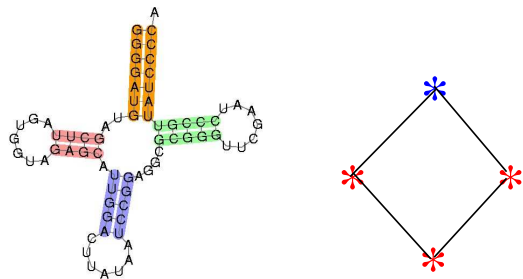


Figure 4: (Left) Example of RNA secondary structure diagram. (Right) The corresponding RNA graph. Node labels are indicated by color.

U). The structure of an RNA is often represented as a secondary structure diagram (Figure 4, left). A successive chain of hybridized pairs is called a *stem*. For example, stems are highlighted in Figure 4, left. The aim of RNA graphs is to represent topological relationships of stems, not individual nucleotides. As shown in Figure 4, right, one stem corresponds to a node and two nodes are connected by an edge if the stems are linked by an intermediate chain of nucleotides. A node can have a self-loop edge, but, due to the restriction of our graph mining algorithm, the self-loop is encoded as a vertex label. Namely, the node is labeled as red, if it has a self-loop, and blue otherwise.

Table 1 shows the experimental results for $\alpha = 0.01, 0.1, 1, 10$. These results were similar. The number of features m is altered from 50 to 5000. The accuracy was evaluated by *cluster purity* [17], where high purity implies that the members of a cluster belong to the same

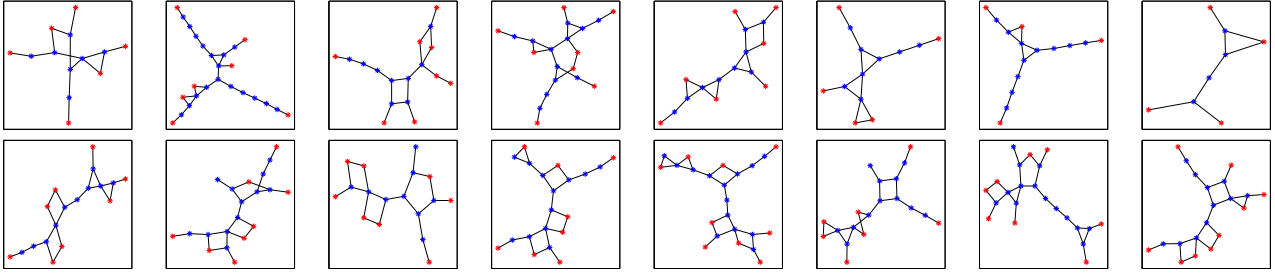


Figure 3: Examples of RNA graphs from Intron-GP-I (top row) and RNaseP-bact-a (bottom row).

true category. The number of clusters is determined by classifying each example to the cluster with highest posterior probability and counting the number of non-empty clusters. Both of the proposed and baseline methods work well if the number of features is more than 500, but the baseline method fails otherwise. In both methods, we observed the tendency that the number of clusters increases along with the number of features. It makes sense, because additional features can reveal detailed structure of the data.

Figure 5 illustrates some of the obtained clusters. They are annotated by *characteristic patterns*, i.e., the patterns with largest θ_{lk} . As seen from this example, graph clusters are difficult to interpret. To understand why the algorithm produced such clusters, annotations by patterns offers a great help to users. On the other hand, similarity-based methods cannot offer such annotations.

5.2 Generalization Performance To evaluate generalization performance, a graph dataset is split into training and test examples. A DP mixture is learned from training examples and the log-likelihood of test examples $\log p_A(\mathbf{x}|\Theta, \Theta_0, \mathbf{v})$ is computed based on the MAP estimates of parameters π, θ_{lk} and θ_{0k} . It is difficult to compute the likelihood (3.12) directly, because the sum over $k \notin A$ assumes the whole feature space is available. Instead, we evaluate

(5.22)

$$\frac{p_A(\mathbf{x}|\Theta, \Theta_0, \mathbf{v})}{p_0(\mathbf{x}|\Theta_0)} = \sum_{l=1}^T \pi_l \prod_{k \in A} \theta_{lk}^{x_k} (1 - \theta_{lk})^{1-x_k} \Big/ \prod_{k \in A} \theta_{0k}^{x_k} (1 - \theta_{0k})^{1-x_k}.$$

where $p_0(\mathbf{x}|\Theta_0) = \prod_{k=1}^d \theta_{0k}^{x_k} (1 - \theta_{0k})^{1-x_k}$ does not depend on cluster assignment or selected features. In this experiment, we used the CPDB dataset containing 683 chemical compounds [4], and the parameters are fixed as $\alpha = 1, a, b = 1, T = 20$.

Based on 10-fold cross validation, the log-likelihood for $m = 50, 100, 1000, 5000$ was computed for both methods. The result is shown in Figure 6 (left). Our method has significantly larger likelihood, confirming that our feature space yields better generalization performance. Figure 6 (right) shows the comparison in terms of free energy. Due to the same reason as (5.22), we used the shifted version of free energy $\hat{\mathcal{F}} = \mathcal{F} - \sum_{i=1}^d D[q_{0k}(\theta_{0k}), p_{0k}(\theta_{0k})]$. Clearly our clusters always have smaller free energy.

The CPDB dataset has two categories (i.e., the chemical compounds with/without mutagenicity), so it is possible to evaluate the classification accuracy as well. However, it turned out that the purity was much poorer than the RNA dataset. This result seems reasonable, because mutagenicity prediction is a very difficult task even in supervised settings [4].

5.3 Computational Cost Table 2 shows the number of nodes in the DFS code tree (i.e., tree size) and overall computational time for clustering. It is understood that our pruning condition worked well to keep the number of nodes rather small. For the larger CPDB dataset, clustering took approximately 9 minutes in maximum, still in a reasonable level. Like other graph mining methods, our algorithm is NP-hard with respect to the pattern size. However, like gspan, the time complexity is empirically linear with respect to the number of training graphs [15].

5.4 Graph Kernels Graph kernels [5] typically derive the similarity of two graphs via random walking. Each graph is represented by a probability distribution of label paths, and the kernel is computed by taking the inner product of two probability vectors. Naturally, graph kernels can also be used to graph clustering. Although there have been no DP-mixture methods using kernels directly, one can embed all examples to a low-dimensional space by the kernel PCA map [10] and apply the DP Gaussian mixture [8]. We have applied

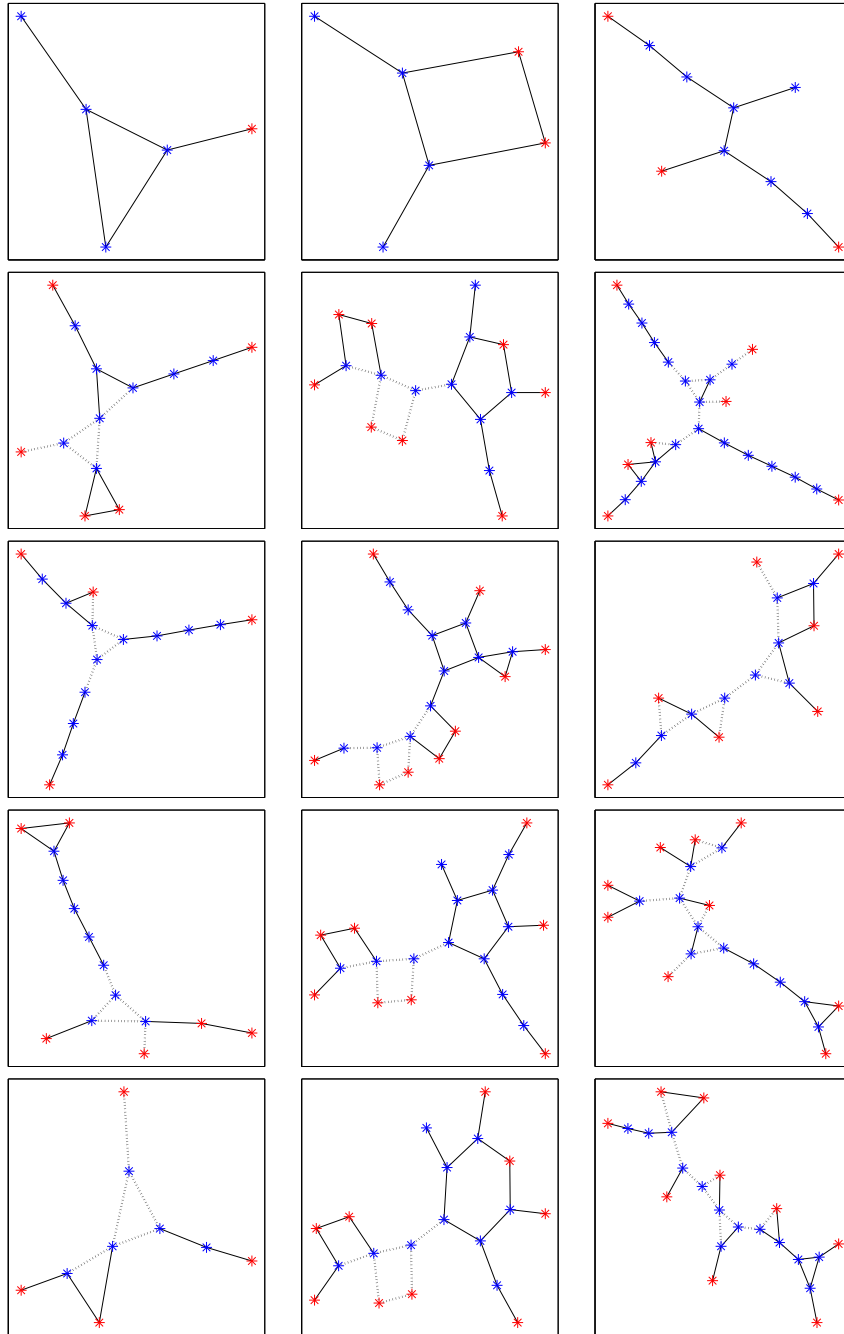


Figure 5: Obtained clusters from the RNA dataset ($m = 1000$). Each column corresponds to a cluster. Characteristic patterns are shown in the uppermost row and their projections are drawn in broken lines.

Table 1: Clustering results in the RNA dataset with different α .

		$\alpha = 0.01$				
#features		50	100	500	1000	5000
Proposed	#cluster	5	7	7	11	11
	purity	87.5	96.2	93.8	93.8	90.0
Frequent Mining	#cluster	3	5	7	9	12
	purity	73.8	73.8	93.8	93.8	88.8

		$\alpha = 1$				
#features		50	100	500	1000	5000
Proposed	#clusters	6	7	7	11	11
	purity	92.5	96.2	93.8	93.8	90.0
Frequent Mining	#clusters	3	5	7	9	12
	purity	73.8	73.8	93.8	93.8	88.8

		$\alpha = 0.1$				
#features		50	100	500	1000	5000
Proposed	#cluster	5	7	7	11	11
	purity	87.5	96.2	93.8	93.8	90.0
Frequent Mining	#cluster	3	5	7	9	12
	purity	73.8	73.8	93.8	93.8	88.8

		$\alpha = 10$				
#features		50	100	500	1000	5000
Proposed	#cluster	6	7	7	11	11
	purity	92.5	96.2	93.8	93.8	90.0
Frequent Mining	#cluster	4	4	7	9	12
	purity	73.8	73.8	93.8	93.8	88.8

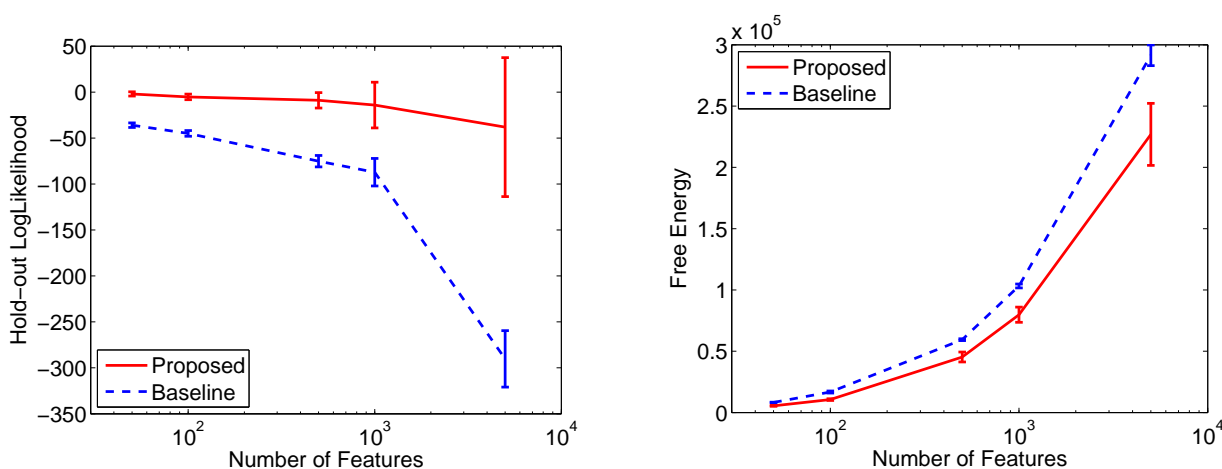


Figure 6: (Left) Average log-likelihood per test example in the CPDB dataset. (Right) Free energy for training data. The lower is the better for the free energy.

marginalized graph kernels [5] to the RNA dataset. The termination probability of random walking was set to 0.1, as it was the best performing parameter in [12]. Actual kernel matrix is shown in Figure 7. In the DP Gaussian mixture model, we have tried a variety of prior distribution with $\alpha = 0.01, 0.1, 1, 10$, but in all cases, only one cluster is detected. Therefore, the graph kernels could not detect underlying class structure at all. This result agrees with the report in [12]: the ROC score of the EM-based clustering was as low as 0.531.

The reason of poor performance of graph kernels is explained as follows: Graph kernels work well, only if the variation of node labels is rich enough. However, if there is only one node label (say 'A'), random walk produces monotonic label paths like 'AAA'. In such a case, the kernel is always one (after normalization) regardless of the topology of compared graphs. In the RNA dataset, the situation is not that bad, but similar,

i.e., we have only two kinds of node labels. As a result, all the kernel values are close to one, and the underlying class structure cannot be detected. On the other hand, graph mining methods use relatively large subgraphs (e.g., 10 nodes) to represent a graph. So they are good at capturing subtle difference in topology. This result does not mean that graph mining methods are always better than graph kernels. Rather, they are complementary to each other. If labels are descriptive enough, one does not need to adopt graph mining methods that require more computational cost.

6 Conclusion

We have presented a method to apply DP mixture clustering to graph data. Although we focused on a particular model, our methodology adopted here is applicable in general: 1) Develop a probabilistic model with feature selection ability, 2) design a tree

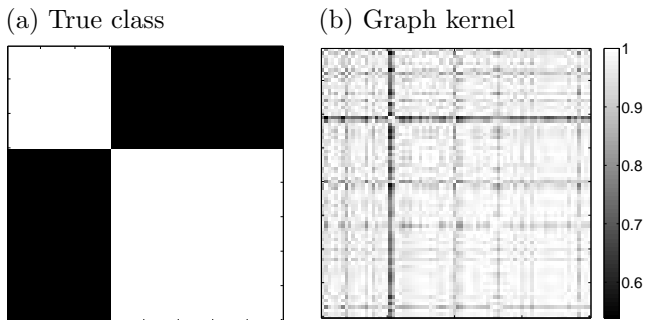


Figure 7: Kernel matrix by marginalized graph kernels for the RNA dataset.

pruning condition on the DFS code tree. Moreover, since all substructure mining algorithms share the same structure (i.e., tree-shaped search space) basically, our methodology is directly applicable to other types or data, such as trees, sequences and itemsets.

References

- [1] D.M. Blei and M.I. Jordan. Variational inference for Dirichlet process mixtures. *Bayesian Analysis*, 1(1):121–144, 2005.
- [2] B. Bringmann, A. Zimmermann, L. D. Raedt, and S. Nijssen. Don’t be afraid of simpler patterns. In *10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 55–66, 2006.
- [3] H.Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *23rd International Conference on Data Engineering*, 2007.
- [4] C. Helma, T. Cramer, S. Kramer, and L.D. Raedt. Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *J. Chem. Inf. Comput. Sci.*, 44:1402–1411, 2004.
- [5] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In T. Faucett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning*, pages 321–328, Menlo Park, CA, AAAI Press, 2003.
- [6] J. Kazius, S. Nijssen, J. Kok, and T. Bäck A.P. Ijzerman. Substructure mining using elaborate chemical representation. *J. Chem. Inf. Model.*, 46:597–605, 2006.
- [7] S. Kim, M.G. Tadese, and M. Vannucci. Variable selection in clustering via Dirichlet mixture models. *Biometrika*, 93(4):877–893, 2006.
- [8] K. Kurihara, M. Welling, and N. Vlassis. Accelerated variational Dirichlet process mixtures. In *NIPS 19*, 2007.
- [9] A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.*, 13:353–362, 1983.
- [10] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.
- [11] J. Sethuraman. A constructive definition of Dirichlet priors. *Statistica Sinica*, 4:639–650, 1994.
- [12] K. Tsuda and T. Kudo. Clustering graphs by weighted substructure mining. In W.W. Cohen and A. Moore, editors, *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 953–960. ACM Press, 2006.
- [13] F. Valente and C. Wellekens. Variational Bayesian feature selection for Gaussian mixture models. In *ICASSP’04*, volume 1, pages 513–516, 2004.
- [14] T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explorations*, 5(1):59–68, 2003.
- [15] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, pages 721–724. IEEE Computer Society, 2002.
- [16] X. Yan and J. Han. gSpan: graph-based substructure pattern mining. Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, 2002.
- [17] Y. Zhao and G. Karypis. Criterion functions for document clustering: Experiments and analysis. Technical Report #01-40, Department of Computer Science, University of Minnesota, 2001.

A Simpler Truncation

In this section, We derive our free energy in (2.6). First of all, we briefly review the free energy of [8]. Then, we derive ours.

Without any constraints, $\mathcal{F} \equiv \lim_{L \rightarrow \infty} \mathcal{F}_L$ is,

$$(1.23) \quad \mathcal{F} = \sum_{l=1}^{\infty} \left\{ D[q_{v_l}(v_l), p_{v_l}(v_l)] + \sum_{k=1}^d D[q_{\theta_{lk}}(\theta_{lk}), p_{\theta_{lk}}(\theta_{lk})] \right\} + \sum_{i=1}^n \mathbb{E}_q[\log q_{y_i}(y_i)] - \mathbb{E}_q[\log p(\mathbf{x}_i, y_i | \Theta, \mathbf{v})].$$

This free energy is intractable due to infinite sum. Kurihara et al.[8] have assumed

$$(1.24) \quad q_{v_l}(v_l) = p_{v_l}(v_l) \text{ for } l > T$$

$$(1.25) \quad q_{\theta_{lk}}(\theta_{lk}) = p_{\theta_{lk}}(\theta_{lk}) \text{ for } l > T.$$

Clearly, the KL divergence $D[q_{v_l}(v_l), p_{v_l}(v_l)]$ and $D[q_{\theta_{lk}}(\theta_{lk}), p_{\theta_{lk}}(\theta_{lk})]$ for $l > T$ become zero. Thus, we can replace $\sum_{l=1}^{\infty}$ in (1.23) with $\sum_{l=1}^T$. Although $\mathbb{E}_q[\log q_{y_i}(y_i)] = \sum_{l=1}^{\infty} q_{y_i}(y_i) \log q_{y_i}(y_i)$ still contains an infinite sum, Kurihara et al. have shown it has an analytical solution.

Table 2: Size of the DFS code tree and computational time. The time is evaluated on a standard PC with 3GHz CPU and 1GB memory.

Number of Features		50	100	500	1000	5000
RNA	Tree Size	8317	8440	8947	9066	10236
	Time (seconds)	14.9	15.9	20.1	27.2	64.3
CPDB	Tree Size	23569	26765	35838	39731	55790
	Time (seconds)	59.9	80.4	128.4	159.7	526.9

To define our free energy, we impose a constraint

$$(1.26) \quad q_{y_i}(y_i = l) = 0 \text{ for } l > T$$

instead of (1.24) and (1.25). But, (1.26) leads to (1.24) and (1.25). Thus, we can replace $\sum_{l=1}^{\infty}$ in (1.23) with $\sum_{l=1}^T$ again. Furthermore, $\mathbb{E}_q[\log q_{y_i}(y_i)]$ is reduced to $\sum_{l=1}^T q_{y_i}(y_i) \log q_{y_i}(y_i)$. Hence, our free energy is tractable as well. The main reason for our constraint instead of (1.24) and (1.25) is the simplicity of q_y , which results in a simplified feature criterion, (3.21).

B Concrete Expression of the Free Energy

In this section, we show the details of derivation from (3.19) to (3.20). From (3.19), we have

$$(2.27) \quad \begin{aligned} \mathcal{F}' = & \sum_{k \in A} \sum_{l=1}^T D[\text{Beta}(\theta_{lk}; a_{lk}, b_{lk}), \text{Beta}(\theta_{lk}; a, b)] \\ & + \sum_{k \notin A} D[\text{Beta}(\theta_{0k}; a_{0k}, b_{0k}), \text{Beta}(\theta_{0k}; a, b)] \\ & - \sum_{i=1}^n \left\{ \mathbb{E}_q[\pi_{y_i}] + \sum_{k \in A} \mathbb{E}_q[\log \text{Bern}(x_{ik}; \theta_{y_i k})] \right. \\ & \left. + \sum_{k \notin A} \mathbb{E}_q[\log \text{Bern}(x_{ik}; \theta_{0k})] \right\} \end{aligned}$$

where $\text{Bern}(\cdot; \cdot)$ is the Bernoulli distribution.

We obtain (3.20) by plugging

$$D[\text{Beta}(\theta; a, b), \text{Beta}(\theta; a', b')]$$

and

$$\mathbb{E}_{\text{Beta}(\theta; a, b)}[\log \text{Bern}(x; \theta)].$$

into (2.27). The KL divergence of beta distributions is

$$\begin{aligned} & D[\text{Beta}(\theta; a, b), \text{Beta}(\theta; a', b')] \\ & = \log B(a', b') - \log B(a, b) \\ & \quad + (a - a')(\Psi(a) - \Psi(a + b)) \\ & \quad + (b - b')(\Psi(b) - \Psi(a + b)), \end{aligned}$$

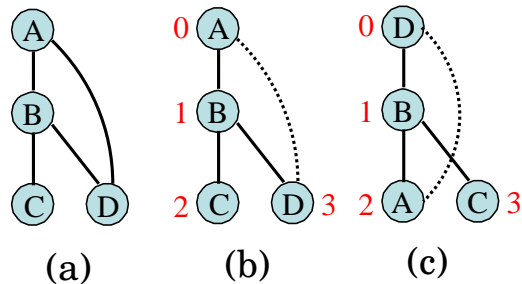


Figure 8: Depth first search and DFS code of graph. (a) A graph example. (b), (c) Two different depth-first searches of the same graph. Red numbers represent the DFS indices. Bold edges and dashed edges represent the forward edges and the backward edges respectively.

where $B(\cdot, \cdot)$ is the beta function, i.e. $B(a, b) = \Gamma(a)\Gamma(b)/\Gamma(a + b)$, and Ψ is the di-gamma function. We also have

$$\begin{aligned} & \mathbb{E}_{\text{Beta}(\theta; a, b)}[\log \text{Bern}(x; \theta)] \\ & = x(\Psi(a) - \Psi(a + b)) + (1 - x)(\Psi(b) - \Psi(a + b)). \end{aligned}$$

C DFS Code Tree

In our algorithm, we need to find the optimal pattern which optimizes a score function. To this end, we need an intelligent way of enumerating all subgraphs of a graph set. This problem is highly nontrivial due to loops: One has to avoid enumerating the same pattern again and again. In this section, we present a canonical search space of graph patterns called *DFS code tree* [16], that enumerates all subgraphs without duplication. In the following, we assume undirected graphs, but it is straightforward to extend the algorithm for directed graphs.

DFS Code The *DFS code* is a string representation of graph G based on depth first search (DFS). According to different starting points and growing edges, there are many ways to perform the search. Therefore, the DFS code of a graph is not unique. To derive a DFS code, each node is indexed from 0 to $n - 1$ according to

the discovery time in the DFS. Denote by E^f the *forward edge* set containing all the edges traversed in the DFS, and by E^b the *backward edge* set containing the remaining edges. Figure 8 show two different indexings of the same graph.

After the indexing, an edge is represented as a pair of indices (i, j) together with vertex and edge labels, $e = (i, j, l_i, l_{ij}, l_j) \in V \times V \times L_V \times L_E \times L_V$, where $V = \{0, \dots, n-1\}$, L_V and L_E are the set of vertex and edge labels, respectively. The index pair is set as $i < j$, if it is an forward edge, and $i > j$ if backward. It is assumed that there are no self-loop edges. To define the DFS code, a linear order \prec_T is defined among edges. For the two edges $e_1 = (i_1, j_1)$ and $e_2 = (i_2, j_2)$, $e_1 \prec_T e_2$ if and only if one of the following statements is true:

1. $e_1, e_2 \in E^f$, and $(j_1 < j_2 \text{ or } i_1 > i_2 \wedge j_1 = j_2)$
2. $e_1, e_2 \in E^b$, and $(i_1 < i_2 \text{ or } i_1 = i_2 \wedge j_1 < j_2)$.
3. $e_1 \in E^b, e_2 \in E^f$, and $i_1 < j_2$.
4. $e_1 \in E^f, e_2 \in E^b$, and $j_1 \leq i_2$.

The DFS code is a sequence of edges sorted according to the above order.

Minimum DFS Code Since there are many possible DFS codes, it is necessary to determine the minimum DFS code as a canonical representation of the graph. Let us define a linear order for two DFS codes $\alpha = (a_0, \dots, a_m)$ and $\beta = (b_0, \dots, b_n)$. By comparing the vertex and edge labels, we can easily build a lexicographical order of individual edges a_i and b_j . Then, the *DFS lexicographic order* for the two codes is defined as follows: $\alpha < \beta$ if and only if either of the following is true,

1. $\exists t, 0 \leq t \leq \min(m, n), a_k = b_k$ for $k < t, a_t < b_t$.
2. $a_k = b_k$ for $0 \leq k \leq m$ and $m \leq n$.

Given a set of DFS codes, the minimum code is defined as the smallest one according to the above order.

Right Most Extension As in most mining algorithm, we form a tree where each node has a DFS code, and the children of a node have the DFS codes corresponding to the supergraphs. The tree is generated in a depth-first manner and the generation of child nodes of a node is done according to the right most extension [16]. Suppose a node has the DFS code $\alpha = (a_0, a_1, \dots, a_n)$ where $a_k = (i_k, j_k)$. The next edge a_{n+1} is chosen such that the following conditions are satisfied:

1. If a_n is a forward edge and a_{n+1} is a forward edge, then $i_{n+1} \leq j_n$ and $j_{n+1} = j_n + 1$.
2. If a_n is a forward edge and a_{n+1} is a backward edge, then $i_{n+1} = j_n$ and $j_{n+1} < i_n$.

3. If a_n is a backward edge and a_{n+1} is a forward edge, then $i_{n+1} \leq i_n$ and $j_{n+1} = i_n + 1$.
4. If a_n is a backward edge and a_{n+1} is a backward edge, then $i_{n+1} = i_n$ and $j_n < j_{n+1}$.

For every possible a_{n+1} , a child node is generated and the extended DFS code (a_0, \dots, a_{n+1}) is stored. The extension is done such that the extended graph is included in at least one graph in the database.

DFS Code Tree The *DFS code tree*, denoted by \mathbb{T} , is a tree-structure whose node represents a DFS code, the relation between a node and its child nodes is given by the right most extension, and the child nodes of the same parent is sorted in the DFS lexicographic order.

It has the following completeness property. Let us remove from \mathbb{T} the subtrees whose root nodes have non-minimum DFS codes, and denote by \mathbb{T}_{min} the reduced tree. It is proven that all subgraphs of graphs in the database are still included in \mathbb{T}_{min} [16]. This property allows us to prune the tree as soon as a non-minimum DFS code is found. The minimality of the DFS code is checked in each node generation, and the tree is pruned if it is not minimum. This minimality check is basically done by exhaustively enumerating all DFS codes of the corresponding graph. Therefore, the computational time for the check is exponential to the pattern size. Techniques to avoid the total enumeration are described in Section 5.1 of [16], but still it is the most time consuming part of the algorithm.