Inferring a graph from path frequency*

Tatsuya Akutsu
Bioinformatics Center, Institute for Chemical Research, Kyoto University,
Uji, Kyoto 611-0011, Japan.

Daiji Fukagawa
Faculty of Culture and Information Science, Doshisha University,
Kyoto 610-0394, Japan.

Jesper Jansson
Ochanomizu University, Tokyo 112-8610, Japan.

Kunihiko Sadakane
National Institute of Informatics, Tokyo 101-8430, Japan.

Abstract

This paper considers the problem of inferring a graph from the numbers of occurrences of vertex-labeled paths, which is closely related to the pre-image problem for graphs: to reconstruct a graph from its feature space representation. It is shown that both exact and approximate versions of the problem can be solved in polynomial time in the size of an output graph by using dynamic programming algorithms if the graphs are trees whose maximum degree is bounded by a constant and the lengths of given paths and alphabet size are bounded by constants. On the other hand, it is shown that this problem is strongly NP-hard even for trees of bounded degree if the maximum length of paths is not bounded. The problem of inferring a string from the number of occurrences of fixed size substrings is also studied.
keywords: Kernel method; Graph algorithms; Feature Vector; Pre-image

1 Introduction

Kernel methods have become a standard tool in machine learning and have been applied to various areas [24, 25], which include bioinformatics and chemoinformatics. In order to apply kernel methods to target problems, it is usually required to develop a mapping from the set of objects in the target problem to a *feature space* (i.e., each object is transformed to a vector of reals) and a kernel function is defined as an inner product between two *feature vectors*. For instance, in the *spectrum kernel* method [17], each biological sequence is mapped to a frequency vector of fixed length substrings (i.e. frequency of n-grams). In some cases, a feature space can be an infinite dimensional space (Hilbert space) and the kernel trick is applied for efficient computation of the value of a kernel function without explicitly computing feature vectors [25].

Recently, a new approach was proposed for designing and/or optimizing objects using kernel methods [4, 5]. In this approach, a desired object is computed as a point in the feature space

*A preliminary version of this paper [1] appeared in Proc. 16th Annual Symposium on Combinatorial Pattern Matching, 2005.

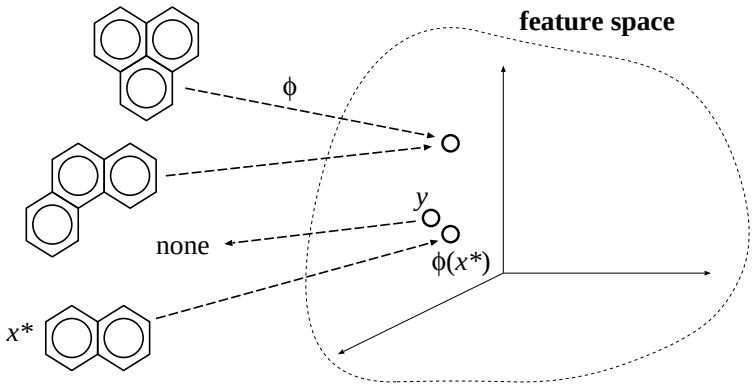


Figure 1: Inference of a graph from a feature vector.

using suitable objective function and optimization technique and then the point is mapped back to the input space, where this mapped back object is called a *pre-image*. Let ϕ be a mapping from an input space $\mathcal{G}$ to a feature space $\mathcal{F}$. Then, the problem is, given a point y in $\mathcal{F}$, to find a pre-image x in $\mathcal{G}$ such that $y = \phi(x)$. It should be noted that ϕ is not necessarily injective or surjective. If ϕ is not surjective, we should compute the approximate pre-image x^* for which the distance between y and $\phi(x)$ is minimized (see Fig. 1):

$$x^* = \arg\min_x \ \ dist(y, \phi(x)).$$

Bakir, Weston and Schölkopf proposed a method to find pre-images in a general setting by using Kernel Principal Component Analysis and regression [4]. Bakir, Zien and Tsuda developed a stochastic search algorithm to find pre-images for graphs [5]. It should be noted that the pre-image problem for graphs is very important from a practical viewpoint because it has potential applications to drug design [5]. For example, suppose that we have two chemical compounds x and y having different functions and we want to design a new chemical compounds having both functions. In such a case, we may develop a new compound by computing a pre-image of the middle point of feature vectors corresponding to x and y (i.e., a pre-image of $(\phi(x) + \phi(x))/2$) [5]. For another example, suppose that we have a potential function for a feature space which reflects some chemical activity and may be obtained from training data using some regression technique. Then, we may develop a chemical compound having more activity by computing a pre-image of a feature vector with maximal or maximum potential score. Indeed, several studies have been done for designing molecules with optimal values using heuristic methods (e.g., genetic algorithms) [20, 28] though they did not use kernel methods. The pre-image problem has another potential application to machine learning. Recent findings in machine learning allow to capture a distribution of points by its mean in feature space [27, 29]. Therefore, by solving the pre-image problem, we may be able to find a graph that represents a whole distribution of graphs.

As mentioned above, several studies have been done on the graph pre-image problem. However, all of the above approaches are statistical, stochastic or heuristic. Therefore, it seems that these do not lead to polynomial time algorithms for solving pre-image problems exactly. In this paper, we study algorithmic aspects of this pre-image problem. Moreover, we focus on pre-image problems for feature spaces defined by frequency of distinct vertex-labeled paths because this type of feature space has been successfully used for classification of biological sequences [17],

glycan structures [13], and chemical compounds [15, 18]. As mentioned in the above, each biological sequence is transformed to a frequency vector of n-grams in the spectrum kernel [17] where each n-gram corresponds to a path of fixed length. For analysis of glycans, which have tree structures, frequency vectors of paths of length at most 3 are used [13]. In the *marginalized graph kernel* [15], each graph (corresponding to a chemical compound) is transformed to a vector of probabilities of vertex-labeled paths. Though some kernel trick was developed for handling unbounded length paths efficiently, it is known that the marginalized kernel works well even if bounded length paths are used [15]. Similar graph kernels were also proposed by Gärtner et al. [12] and Shervashidze et al. [26] Thus, we only consider the case of bounded length paths. Moreover, we do not consider the probabilities; instead we consider for simplicity the number of occurrences of labeled paths as in the case of the spectrum kernel because it is considered to be a fundamental case (the probability corresponds to the number of occurrences if we consider regular graphs).

We first show that the exact pre-image problem for sequences can be solved in polynomial time in the length of the sequence to be inferred. Though this result is almost trivial from the *Eulerian path approach*, which was originally developed for *sequencing by hybridization* [22], we also show that the approximate pre-image problem for sequences can be solved using a dynamic programming algorithm. Next, we present dynamic programming algorithms for both exact and approximate pre-image problems for trees of bounded degree (i.e., the maximum degree is bounded by a constant), which work in polynomial time in the size of an output graph when a feature vector is defined by frequency of bounded length paths over a finite set of labels, where the length of a path in a graph is defined in this paper as the number of edges on the path. On the other hand, we show that the pre-image problem is strongly NP-hard even for trees of bounded degree if the lengths of paths are not bounded by a constant. Though the algorithms proposed for graphs are not practical and there still remain gaps between the positive and negative results, these results provide new insights into the pre-image problem.

Several related problems have been studied in theoretical computer science. As mentioned above, sequencing by hybridization [22] is almost the same as the pre-image problem for the spectrum kernel. Furthermore, Cortes et al. showed that the pre-image problem for the spectrum kernel can be solved in polynomial time [7, 8][1]. However, extension of the results on the spectrum kernel to graphs is far from trivial, and they did not give a concrete theoretical result on the approximate version of the problem. Graphical degree sequence problems [3], graph inference from walks [19, 23] and the graph reconstruction problem [16] are related to the pre-image problem for graphs. However, to our knowledge, results on these three graph problems are not directly applicable to the pre-image problem for graphs.

After a preliminary version of this paper was published [1], several related results were obtained. Nagamochi showed that the pre-image problem for graphs can be solved in polynomial time if the lengths of paths are at most 1, by using *graph detachment* [21]. However, his algorithm cannot be applied to the cases of longer length paths. Fujiwara et al. [10] and Ishida et al. [14] developed practical branch-and-bound algorithms for the pre-image problem for tree-like structures. Akutsu and Fukagawa studied pre-image problems on graphs for feature vectors based on fragments and developed inference algorithms for outer-planar graphs with some constraints [2]. However, these works are motivated by and/or are based on the work

[1]The results on the pre-image problem for spectrum kernel in this paper were obtained independently of their work. Indeed, a preliminary conference version of this paper [1] appeared in almost the same time period.

appeared in a preliminary version of this paper [1][2].

2 Problem definitions

Before presenting algorithms and hardness results, we define the problems formally. First, we define the pre-image problem for sequences (i.e., strings). We use feature vectors in the spectrum kernel [17]. Let Σ be an alphabet and Σ^K be the set of strings with length K over Σ. For two strings t and s, $occ(t, s)$ denotes the number of occurrences of substring t in s. Then, the feature vector $\boldsymbol{f}_K(s)$ of level K for s is a $|\Sigma^K|$-dimensional integer vector such that the coordinate indexed by $t \in \Sigma^K$ is $occ(t, s)$. That is, $\boldsymbol{f}_K(s)$ is defined by

$$\boldsymbol{f}_K(s) = (occ(t, s))_{t \in \Sigma^K}.$$

As an example, consider string 00101111 over $\Sigma = \{0, 1\}$. Then, $\boldsymbol{f}_2(s) = (1, 2, 1, 3)$ because $occ(00, s) = 1$, $occ(01, s) = 2$, $occ(10, s) = 1$ and $occ(11, s) = 3$.

If K is large, the number of dimensions of a feature vector will be large (exponential in K). In such a case, many coordinates will have value 0. Thus, when K is not a constant, we assume that a vector is represented in an appropriate way (linear size of a target sequence) so that the coordinates having value 0 are not included in the data structures.

Definition 1 (**SISF**: Sequence Inference from Spectrum Feature) *Given a feature vector $\boldsymbol{v}$ of level K, output a string s satisfying $\boldsymbol{f}_K(s) = \boldsymbol{v}$. If there does not exist such s, output "no solution".*

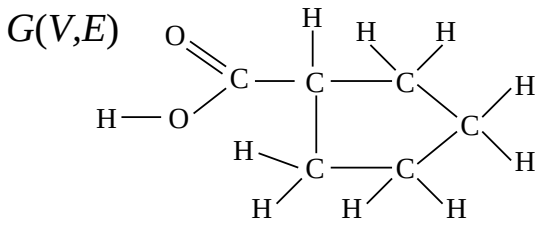
The above definition can be extended to the case of finding the sequence nearest to a given feature vector. Let $dist(\boldsymbol{u}, \boldsymbol{v})$ denote the L_p distance between $\boldsymbol{u}$ and $\boldsymbol{v}$ throughout this paper, where p is an arbitrary fixed positive integer.

Definition 2 (**SISF-M**: Sequence Inference from Spectrum Feature with Minimum Error) *Given a feature vector $\boldsymbol{v}$ of level K, output a string s such that $dist(\boldsymbol{f}_K(s), \boldsymbol{v})$ is the minimum.*

Next, we define pre-image problems for graphs. Let $G(V, E)$ be an undirected vertex-labeled connected graph and Σ be a set of vertex labels. A sequence of vertices $(v_0, v_1, \ldots, v_h)$ of G is called a *path* of length h ($h \geq 0$) if $\{v_i, v_{i+1}\} \in E$ holds for $i = 0, \ldots, h-1$. It should be noted that the same vertex (and the same edge) can appear more than once in the above definition. Since many papers on marginalized graph kernels [15, 18, 24] use this notation, we employ this definition of a path. Let $\Sigma^{\leq k}$ be the set of label sequences (i.e., the set of strings) over Σ whose lengths are between 1 and k. Let $l(v)$ be the label of vertex v. For a path $P = (v_0, \ldots, v_h)$ of G, $l(P)$ denotes the label sequence of P (i.e., $l(P) = l(v_0)l(v_1)\ldots l(v_h)$). It should be noted that the length of $l(P)$ is the length of P plus one (i.e., the length of a path is the number of edges and its label sequence contains length+1 letters). For graph G and label sequence t, $occ(t, G)$ denotes the number of paths P in G such that $l(P) = t$. Then, the feature vector $\boldsymbol{f}_K(G)$ of level K for $G(V, E)$ is an integer vector such that the coordinate index by $t \in \Sigma^{\leq K+1}$ is $occ(t, G)$. That is, $\boldsymbol{f}_K(G)$ is defined by

$$\boldsymbol{f}_K(G) = (occ(t, G))_{t \in \Sigma^{\leq K+1}}.$$

[2]Detailed explanations and/or proofs as well as some additional results are newly given in this paper.



$\mathbf{f}_1(G)$

C	O	H	CC	CO	CH	OC	OO	OH	HC	HO	HH
6	2	10	12	2	9	2	0	1	9	1	0

Figure 2: Example of GIPF for $\Sigma = \{\mathrm{C,O,H}\}$ and $K = 1$. It is to be noted that each edge is counted twice for forward and backward directions. The double bond O=C in the figure is considered to be a single edge in the graph.

For example, consider a star $G(V, E)$ consisting of four vertices where the center vertex has label 0 and the other three vertices have label 1. Then, $\boldsymbol{f}_1(G) = (1, 3, 0, 3, 3, 0)$ because $occ(0, G) = 1$, $occ(1, G) = 3$, $occ(00, G) = 0$, $occ(01, G) = 3$, $occ(10, G) = 3$ and $occ(11, G) = 0$. See Fig. 2 for a more general example.

Our feature vector differs from that of the spectrum kernel in that it counts the number of paths of *all* lengths from 0 to K, as in the case of the marginalized graph kernel [15]. In this paper, we mainly consider for simplicity the case where *tottering paths* (paths for which there exists some i such that $v_i = v_{i+2}$) are not counted in feature vectors because removal of tottering paths does not decrease the prediction accuracy [18]. Therefore, paths mean non-tottering paths unless otherwise stated. However, all the results on graphs in this paper are valid even if tottering paths are taken into account. Since tottering paths are used in several graph kernels [12, 15, 26], we also describe the required modifications for including tottering paths when necessary.

Definition 3 (**GIPF**: Graph Inference from Path Frequency) *Given a feature vector $\boldsymbol{v}$ of level K, output a connected graph $G(V, E)$ satisfying $\boldsymbol{f}_K(G) = \boldsymbol{v}$. If there does not exist such a $G(V, E)$, output "no solution".*

A feature vector $\boldsymbol{v}$ satisfying the above condition is called *realizable*. It is to be noted that $G(V, E)$ is not necessarily uniquely determined (i.e., there may exist multiple graphs G satisfying $\boldsymbol{f}_K(G) = \boldsymbol{v}$). The above definition can be extended to the case of finding the graph nearest to a given feature vector as in Definition 2.

Definition 4 (**GIPF-M**: Graph Inference from Path Frequency with Minimum Error) *Given a feature vector $\boldsymbol{v}$ of level K, output a connected graph $G(V, E)$ such that $dist(\boldsymbol{f}_K(G), \boldsymbol{v})$ is the minimum.*

It is worthy to note that both the number of dimensions of the feature vector and the maximum coordinate value of the feature vector are bounded by a polynomial in the size of the graph if we consider constant K.

3 Algorithms for sequences

In this section, we present algorithms for SISF and SISF-M. Though the algorithm for SISF follows directly from the Eulerian path approach for sequencing by hybridization [22], the algorithm for SISF-M is based on another technique (dynamic programming).

Suppose that a feature vector $\boldsymbol{v}$ of level K over a fixed Σ is given for the SISF problem (i.e., $occ(t, s)$ is given). We construct a directed multi graph $G'(V, E)$ such that $V = \Sigma^{K-1}$ and there exist directed edges (t', t'') with multiplicity $occ(t, s)$ for each $t \in \Sigma^K$, where t' and t'' are the prefix and suffix of t with length $K-1$, respectively. Then, we can see that there exists an Eulerian path in $G'(V, E)$ if and only if there is a solution for the SISF problem. It should be noted that we need not create vertices which are neither prefix or suffix of any substring t such that $occ(t, s) > 0$. Let n be the length of the target sequence (i.e., $n = K - 1 + \sum_t occ(t, s)$). Since the length of each t is K and Σ is fixed, we can construct $G'(V, E)$ in $O(n)$ time using simple appropriate data structure (e.g, trie). Using the result on sequencing by hybridization [22], we have:

Proposition 1 *SISF is solved in $O(n)$ time for a fixed Σ.*

It is to be noted that the same result was obtained in [7]. SISF can be solved in $O(n \log n)$ time for a general alphabet since $O(\log n)$ additional time is required to maintain a trie. Though we may be able to reduce the time complexity to $O(n)$ even for a general alphabet by using some sophisticated data structure (e.g., suffix tree for a general alphabet), we do not consider such a case because $|\Sigma|$ is fixed to 4 or 20 in biological applications.

For the SISF-M problem, we may construct a graph $G'(V, E)$ in the same way. However, $G'(V, E)$ does not necessarily have an Eulerian path. Thus, we should add or delete edges (using the minimum number of additions/deletions) so that the resulting graph has an Eulerian path. It is unclear whether or not such an approach leads to a polynomial time algorithm. Though Cortes et al. proposed some heuristic algorithm for a similar problem, they did not get any concrete theoretical result [8]. Here, we show that SISF-M can be solved in polynomial time in n using a dynamic programming algorithm if K and Σ are fixed.

Theorem 1 *SISF-M is solved in polynomial time in n if K and Σ are fixed.*

Proof. We only show the algorithm for $K = 2$ and a binary alphabet (i.e., $\Sigma = \{0, 1\}$) since extension to the other cases is straightforward. For string $s = s_1 s_2 \cdots s_m$, we define $s[i] = s_i$, $s[i, j] = s_i s_{i+1} \cdots s_j$, and $|s| = m$. We construct a table $F(\cdots)$ defined by

$$F(n_{00}, n_{01}, n_{10}, n_{11}, t) =$$
$$\begin{cases} 1, & \text{if there exists sequence } s \text{ such that } \boldsymbol{f}_2(s) = (n_{00}, n_{01}, n_{10}, n_{11}) \\ & \text{and } s[|s|-1, |s|] = t, \\ 0, & \text{otherwise.} \end{cases}$$

This table can be constructed by the following dynamic programming algorithm.

$F(n_{00}, n_{01}, n_{10}, n_{11}, t) = 1$ **iff**
$F(n_{00} - 1, n_{01}, n_{10}, n_{11}, t') = 1$ **and** $t'[2] = t[1]$ **and** $t = 00$ **or**
$F(n_{00}, n_{01} - 1, n_{10}, n_{11}, t') = 1$ **and** $t'[2] = t[1]$ **and** $t = 01$ **or**
$F(n_{00}, n_{01}, n_{10} - 1, n_{11}, t') = 1$ **and** $t'[2] = t[1]$ **and** $t = 10$ **or**

$F(n_{00}, n_{01}, n_{10}, n_{11} - 1, t') = 1$ **and** $t'[2] = t[1]$ **and** $t = 11$ holds
for some $t' \in \{00, 01, 10, 11\}$,

where it is initialized as

$F(1, 0, 0, 0, 00) = 1$,
$F(0, 1, 0, 0, 01) = 1$,
$F(0, 0, 1, 0, 10) = 1$,
$F(0, 0, 0, 1, 11) = 1$,
$F(n_{00}, n_{01}, n_{10}, n_{11}, t) = 0$, otherwise.

It is straightforward to see that this algorithm correctly computes the entries of F.

Let $\boldsymbol{u} = (n_{00}, n_{01}, n_{10}, n_{11})$ be a given feature vector. Recall that we assumed that the length of the target string is n and thus $n_{00}+n_{01}+n_{10}+n_{11} = n-1$ holds. Since $F(n-1, 0, 0, 0, 00) = 1$ holds for $s = 00\cdots0$ of length n and

$$((n - 1 - n_{00})^p + n_{01}^p + n_{10}^p + n_{11}^p)^{1/p} < 2n$$

holds for any positive integer p, the minimum error for SISF-M is less than $2n$. We can also see that if $dist(\boldsymbol{u}, \boldsymbol{v}) < 2n$ holds for any $\boldsymbol{v} = (n'_{00}, n'_{01}, n'_{10}, n'_{11})$, $n'_{ij} < 3n$ holds for all i, j. Therefore, it is enough to consider a table of size $3n \times 3n \times 3n \times 3n \times 4$, which is $O(n^4)$. Furthermore, such a table can also be constructed in $O(n^4)$ time since each entry can be calculated in constant time. Therefore, by examining all the entries of such a table, we can find $\boldsymbol{v}$ minimizing $dist(\boldsymbol{f}_2(s), \boldsymbol{v})$ in $O(n^4)$ time. The required string can be obtained by using the standard *traceback* technique for reconstructing an optimal solution from a dynamic programming table.

In general, $F(\cdots)$ is a $(|\Sigma|^K + 1)$-dimensional table with size $O((3n)^{|\Sigma|^K} \cdot |\Sigma|^K)$. It is still a polynomial in n for fixed Σ and K. Furthermore, each entry can also be calculated in constant time and $n_{00}^p+n_{01}^p+n_{10}^p+n_{11}^p$ can also be calculated in constant time for fixed Σ, K and p, where we assume that each of addition and multiplication can be done in constant time. Therefore, the theorem holds. □

4 Algorithms for trees

In this section, we present dynamic programming algorithms for inference of trees from feature vectors of constant levels. They do not work in polynomial time with respect to the size of a feature vector because a feature vector may be represented in $O(\log n)$ size, where n is the number of vertices of the graph (i.e., n is the sum of frequencies of paths of length 0). However, they work in pseudo polynomial time (i.e., in time which is polynomial in n). Considering such algorithms is quite reasonable because these work in polynomial time with respect to the size of an output graph.

4.1 Algorithm for level 1 feature vectors

We begin with the case of inference of trees from feature vectors of level 1. It is to be noted that the following result has already been improved by Nagamochi [21]: he showed that GIPF for $K = 1$ can be solved in polynomial time for both trees and general graphs with general alphabet. However, his results or methods cannot be applied to the case of $K > 1$. We present the following result because it is useful for understanding the algorithm for a more general (but

bounded degree) case to be shown in Theorem 3. It is also to be noted that we need not consider tottering paths here because there are no tottering paths in the case of $K = 1$.

Theorem 2 *GIPF for trees is solved in polynomial time in n for $K = 1$ and a fixed alphabet.*

Proof. We only show the algorithm for the case of the binary alphabet, where extension to an arbitrary fixed alphabet is straightforward. As in Theorem 1, we construct a dynamic programming table $F(\cdots)$. In this case, $F(\cdots)$ is defined by

$$F(n_0, n_1, n_{00}, n_{01}, n_{10}, n_{11}) = \begin{cases} 1, & \text{if there exists tree } T \text{ such that } \boldsymbol{f}_1(T) = (n_0, n_1, n_{00}, n_{01}, n_{10}, n_{11}), \\ 0, & \text{otherwise.} \end{cases}$$

This table can be constructed by the following dynamic programming procedure where the initialization part is straightforward.

$F(n_0, n_1, n_{00}, n_{01}, n_{10}, n_{11}) = 1$ **iff**
$F(n_0 - 1, n_1, n_{00} - 2, n_{01}, n_{10}, n_{11}) = 1$ **or**
$F(n_0 - 1, n_1, n_{00}, n_{01} - 1, n_{10} - 1, n_{11}) = 1$ **or**
$F(n_0, n_1 - 1, n_{00}, n_{01} - 1, n_{10} - 1, n_{11}) = 1$ **or**
$F(n_0, n_1 - 1, n_{00}, n_{01}, n_{10}, n_{11} - 2) = 1.$

The first, second, third and fourth conditions correspond to the cases where a new vertex labeled 0 is connected to an existing vertex labeled 0, a new vertex labeled 0 is connected to an existing vertex labeled 1, a new vertex labeled 1 is connected to an existing vertex labeled 0, and a new vertex labeled 1 is connected to an existing vertex labeled 1, respectively.

The correctness of the algorithm follows from the fact that any tree can be constructed incrementally by adding vertices (leaves) one by one. Since a new vertex can be connected with any vertex in the current tree, different from Theorem 1, we need not maintain the last part of a fragment. The required tree (if exists) can be obtained by using a traceback procedure. Since the value of each coordinate of a feature vector is $O(n)$ where $n = n_0 + n_1$, the table size is $O(n^6)$ and thus the computation time is $O(n^6)$.

In general, we should construct a $(|\Sigma|+|\Sigma|^2)$-dimensional table of size $O(n^{|\Sigma|+|\Sigma|^2})$. However, it is still polynomial in n for any fixed Σ. □

Note that a generate-and-test approach (enumerating all trees of size n and checking whether each tree T satisfies $\boldsymbol{f}_K(T) = \boldsymbol{v}$) does not yield a (pseudo) polynomial time algorithm because the number of possible trees is not bounded by a polynomial in n.

The algorithm above can be modified for GIPF-M. We only need to consider the table such that each coordinate value is $O(n)$ because any tree of size $O(n)$ can correspond to some element in that table. Furthermore, for any n, there exists at least one tree of size n. Therefore, as in Theorem 1, we can find the feature vector $\boldsymbol{f}_K(T)$ closest to $\boldsymbol{v}$ in polynomial time by examining all vectors in the table of polynomial size.

Corollary 1 *GIPF-M for trees is solved in polynomial time in n for $K = 1$ and a fixed alphabet.*

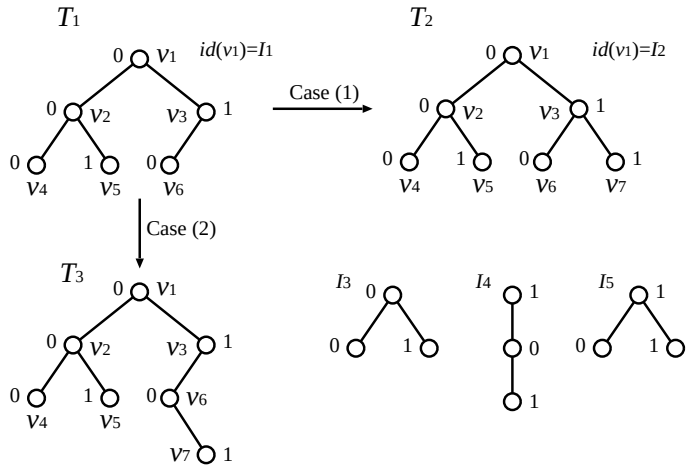


Figure 3: Examples for the dynamic programming procedure for GIPF.

4.2 Algorithm for level K feature vectors

We extend the above algorithm to cases of $K > 1$, where K is a constant and we only consider trees over a fixed Σ whose maximum degree is bounded by a constant D. This extension is not straightforward and is somewhat involved.

Though we do not consider directed trees, we will treat an undirected tree as if it were a rooted tree. Let r be the root of a tree T. The *depth* (denoted by $d(v)$) of a vertex $v \in T$ is the length of the (shortest) path from r to v. The *height* of a tree (denoted by $d(T)$) is the depth of the deepest vertex. The *size* of a tree (denoted by $|T|$) is the number of vertices in T.

For each vertex $v \in T$, $T(v)$ denotes the subtree of T induced by v and its descendants. In the following, we only consider $T(v)$s such that v is at depth $d(T) - K$. Thus, we only consider subtrees of height at most K. $ID(v)$ denotes the *signature* (i.e., canonical labeling in [9]) of v where the signature is an integer number of value $O(|T(v)|)$ such that $ID(v) = ID(v')$ if and only if $T(v)$ is isomorphic to $T'(v')$. Since we consider constant K and trees of bounded degree, there are $O(1)$ different $ID(v)$s, the value of each $ID(v)$ is $O(1)$, and $ID(v)$ can be computed in $O(1)$ time per v.

For each tree T, $E(d, id)$ denotes the number of vertices v such that $d(v) = d$ and $ID(v) = id$. Let $\boldsymbol{e}$ denote the vector consisting of $E(d, id)$ for $d = d(T) - K$ and for all possible id. Let $\boldsymbol{g}_K(T)$ denote $\boldsymbol{e}$ for T. It should be noted that the number of dimensions of $\boldsymbol{e}$ is bounded by a constant since there exist $O(1)$ different signatures.

Then, we construct a table $F(\boldsymbol{v}, \boldsymbol{e}, d)$ defined by

$$F(\boldsymbol{v}, \boldsymbol{e}, d) = \begin{cases} 1, & \text{if there exists a tree } T \text{ such that } \boldsymbol{f}_K(T) = \boldsymbol{v}, \boldsymbol{g}_K(T) = \boldsymbol{e} \\ & \text{and } d(T) = d, \\ 0, & \text{otherwise.} \end{cases}$$

It is to be noted that the size of trees is uniquely determined from $\boldsymbol{v}$ though trees may not be uniquely determined. It should also be noted that we can not maintain the whole structure of

a tree, instead we maintain a feature vector, subtrees (in the form of $ID(v)$s) rooted at depth $d-K$, and the depth of a tree.

Here, we explain $F(\boldsymbol{v}, \boldsymbol{e}, d)$ using examples in Fig. 3, where we consider the case of $K = 2$ and $\Sigma = \{0, 1\}$. $F(\boldsymbol{v}, \boldsymbol{e}, d)$ has the following form

$$F(n_0, n_1,\ n_{00}, \ldots, n_{11},\ n_{000}, \ldots, n_{111},\ E(d(T)-2, I_1), \ldots, E(d(T)-2, I_H),\ d(T)),$$

where H is the number of different possible signatures. It should be noted that 5 relevant signatures are only shown in Fig. 3, where I_5 is to be used later. Then, T_1 corresponds to

$$F(4, 2, \quad 4, 3, 3, 0, \quad 1, \ldots, 0, \quad 1, 0, 0, 0, 0, \ldots, 0, \quad 2),$$

T_2 corresponds to

$$F(4, 3, \quad 4, 3, 3, 2, \quad 1, \ldots, 0, \quad 0, 1, 0, 0, 0, \ldots, 0, \quad 2),$$

and T_3 corresponds to

$$F(4, 3, \quad 4, 4, 4, 0, \quad 1, \ldots, 0, \quad 0, 0, 1, 1, 0, \ldots, 0, \quad 3).$$

Construction of the table is done in an incremental manner as in the case of $K = 1$. However, in this case, we only add a new vertex at depth either d or $d+1$ (see also Fig. 4). It should be noted that any tree can be constructed in this manner.

First, we consider the case of adding a new vertex at depth $d = d(T)$ to a tree T (see also Case (1) of Fig. 3). For each subtree $T(v)$ corresponding to an element of $\boldsymbol{e}$ such that $E(d-K, ID(v)) > 0$, we consider all ways of appending a new vertex w having label $l \in \Sigma$ to $T(v)$. It is enough to consider one subtree with the same signature in the case of $E(d-K, ID(v)) > 1$ since addition of w only affects paths within $T(v)$. Then, we can see that the number of ways of appending a new vertex is bounded by a constant per $T(v)$ because the number of possible subtrees (of height K with bounded degree) is bounded by a constant. Moreover, addition of w affects at most two elements of $\boldsymbol{e}$. Furthermore, the change of $\boldsymbol{f}_K(T)$ is of a constant size and is determined only by w, $T(v)$, and the position of $T(v)$ to which w is connected. For example, in Case (a) of Fig. 3, $E(0, I_1)$ decreases from 1 to 0 and $E(0, I_2)$ increases from 0 to 1, and the difference between $\boldsymbol{f}_K(T_1)$ and $\boldsymbol{f}_K(T_2)$ is determined only by I_1 and I_2. Since there are a constant number of signatures, we can pre-compute all the required changes of signatures and $\boldsymbol{f}_K(T)$s for all possible ways of addition of a new vertex in constant time. Suppose that $F(\boldsymbol{v}, \boldsymbol{e}, d) = 1$ holds and $(\boldsymbol{v}, \boldsymbol{e}, d)$ changes to $(\boldsymbol{v}', \boldsymbol{e}', d)$ by addition of w to some position of $T(v)$. Then, $F(\boldsymbol{v}', \boldsymbol{e}', d)$ is set to 1 (initially, all entries of the table are set to 0). This update of the table can be done in constant time per $(\boldsymbol{v}, \boldsymbol{e}, d)$ since there is a constant number of ways of addition of a new vertex.

Next, we consider the case of adding a new vertex at depth $d(T)+1$ to a tree T of size n (see also Case (2) in Fig. 3). As above, we consider all ways of appending a new vertex. Unlike the above, all entries of $\boldsymbol{e}$ may change because the height of the tree changes. For example, in Case (2) of Fig. 3, $E(0, I_1)$ decreases from 1 to 0, and $E(1, I_3)$ and $E(1, I_4)$ are set to 1. For another example, consider a tree T_0 of height 3 such that the root is labeled with 1 and has three children v_1, v_2 and v_3 such that $T(v_1)$ is isomorphic to T_1, and each of $T(v_2)$ and $T(v_3)$ is isomorphic to T_2. Suppose that the subtree isomorphic to T_1 changes to a subtree isomorphic to T_3. Then, $\boldsymbol{e}$ changes from

$$(1, 2, 0, 0, 0, \ldots, 0)$$

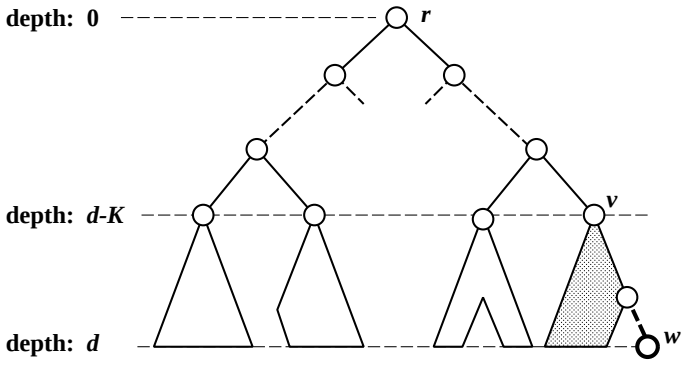


Figure 4: Addition of a new vertex w to tree T. Vector $\boldsymbol{e}$ corresponds to the numbers of appearances of isomorphic subtrees rooted at depth $d-K$. Addition of w only affects the signature of $T(v)$ and paths (of length at most K) in the gray part (i.e., $T(v)$)

to

$$(0, 0, 3, 1, 2, \ldots, 0)$$

because I_1 changes to I_3 and I_4, and I_2 changes to I_3 and I_5. However, this kind of changes can also be computed in $O(n)$ time because there exist a constant number of possible signatures (i.e., different subtrees), the maximum degree is bounded by a constant D and the number of subtrees rooted at each depth is bounded by n. It should be noted that decrease of one $E(d-K, ID(v))$ corresponds to increases of multiple but a constant number of $E(d-K+1, ID(v'))$s. Furthermore, changes of $\boldsymbol{f}_K(T)$ can also be computed in $O(n)$ time since required changes of $\boldsymbol{f}_K(T)$ per signature are bounded by a constant. Therefore, update of the table can be done in $O(n)$ time per addition of a new vertex.

We can fill in the table entries in the order according to the number of vertices in tree T since the number of vertices increases by 1 per addition of a new vertex and this number is obtained from $\boldsymbol{v}$. After initializing all the table entries to be 0, we can fill in the table entries with $d = 0$ by examining all possible trees of bounded degree D and height K, which can be done in $O(1)$ time. The correctness of the algorithm can be seen from the fact that any tree can be obtained by adding vertices one by one at depth d or depth $d+1$, where d is the current depth. Then, we analyze the time complexity. The number of dimensions of table $F(\boldsymbol{v}, \boldsymbol{e}, d)$ is bounded by a constant. The value of each coordinate is $O(n)$ since there exist $O(n \cdot \sum_{k=0}^{K}(D-1)^k) = O(n \cdot (D-1)^{K+1})$ non-tottering paths and there exist $O(n)$ subtrees rooted at each depth. Therefore, the size of table $F(\boldsymbol{v}, \boldsymbol{e}, d)$ is bounded by a polynomial of n. As seen above, $O(n)$ time is required per update of a table entry. Thus, the algorithm works in polynomial time in n. The required tree can be obtained by using the traceback technique. Therefore, we have:

Theorem 3 *GIPF for trees of bounded degree is solved in polynomial time in n if K and Σ are fixed.*

Since there exists a tree of size n for all $n > 0$, we can extend this result to GIPF-M as in Corollary 1.

Corollary 2 *GIPF-M for trees of bounded degree is solved in polynomial time in n if K and Σ are fixed.*

Theorem 3 and Corollary 2 hold also in the tottering case. In that case, we can use the same algorithm except that $F(\boldsymbol{v}, \boldsymbol{e}, d)$ and the rules of update are modified according to the change of the definition of paths. Since we assume that D, K, and Σ are fixed, the number of dimensions of $F(\boldsymbol{v}, \boldsymbol{e}, d)$ is still bounded by a constant. Furthermore, the computation time of update of the table remains constant and $O(n)$ when a new vertex is added at depth $d(T)$ and at depth $d(T) + 1$, respectively. If we allow tottering paths, the number of possible paths is increased from $O(n \cdot (D-1)^{K+1})$ to $O(n \cdot D^{K+1})$. However, it is still $O(n)$ and thus the size of table $F(\boldsymbol{v}, \boldsymbol{e}, d)$ is still bounded by a polynomial of n. Therefore, we can still solve GIPF and GIPF-M in polynomial time in n.

5 Hardness of graph inference

In this section, we show strong NP-hardness results on GIPF, from which the same strong NP-hardness results on GIPF-M follow.

5.1 Hardness of GIPF with $K = 2$

To prove the strong NP-hardness of GIPF for $K = 2$, we provide a *pseudo polynomial-time reduction* [11] from the following problem:

Three-Dimensional Matching (3DM)

Instance: A set $M \subseteq A \times B \times C$, where $A = \{a_1, \ldots, a_q\}$, $B = \{b_1, \ldots, b_q\}$, and $C = \{c_1, \ldots, c_q\}$ and A, B, and C are disjoint sets.

Question: Is there a subset M' of M with $|M'| = q$ such that M' is a matching, i.e., such that for every pair $e_1, e_2 \in M'$ it holds that e_1 and e_2 differ in all coordinates?

3DM is NP-hard, even if restricted to instances which satisfy the *pairwise consistency constraint* (see, e.g., [11]):

> For every $a \in A$, $b \in B$, $c \in C$, if M contains three triples of the form $(a, b, *)$, $(a, *, c)$, and $(*, b, c)$ then $(a, b, c) \in M$.

The reduction from 3DM to GIPF is as follows. Given an arbitrary instance of 3DM which satisfies the pairwise consistency constraint, create an instance of GIPF with $K = 2$, $\Sigma = A \cup B \cup C \cup \{y, z\}$, where y and z are two new symbols not belonging to $A \cup B \cup C$, and a feature vector v of level K over Σ. For any $i, j \in A \cup B \cup C$, denote the number of triples in M that contain i by $s(i)$ and the number of triples in M that contain both i and j by $s(i, j)$. For any $t \in \Sigma^{\leq K+1}$, denote the coordinate in v for t by $v[t]$, and set its value according to the rules below.

- $|t| = 1$:
  $v[i] = 1$ for each $i \in A \cup B \cup C$, $\quad v[y] = |M|$, $\quad v[z] = 1$. (1)

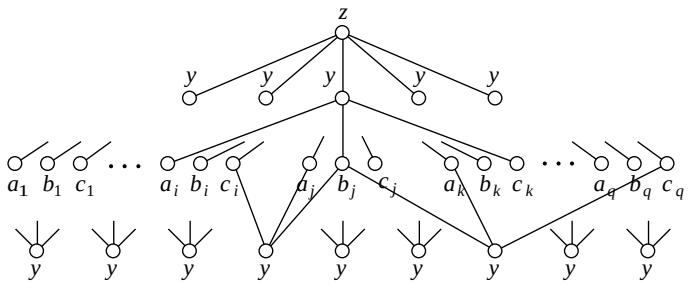


Figure 5: A graph G which realizes v. In this example, (a_i, b_j, c_k), (a_j, b_j, c_i), $(a_k, b_j, c_q) \in M$ with $(a_i, b_j, c_k) \in M'$ but (a_j, b_j, c_i), $(a_k, b_j, c_q) \notin M'$. Note that exactly q of the vertices labeled by y are adjacent to the vertex labeled by z, and the remaining $|M| - q$ are not.

- $|t| = 2$:
  $v[(i, y)] = v[(y, i)] = s(i)$ for each $i \in A \cup B \cup C$, (2)
  $v[(y, z)] = v[(z, y)] = q$, (3)

- $|t| = 3$:
  $v[(i, y, z)] = v[(z, y, i)] = 1$ for each $i \in A \cup B \cup C$, (4)
  $v[(y, i, y)] = s(i) \cdot (s(i) - 1)$ for each $i \in A \cup B \cup C$, (5)
  $v[(i, y, j)] = v[(j, y, i)] = s(i, j)$ for each $i, j \in A \cup B \cup C$, (6)
  $v[(y, z, y)] = q \cdot (q - 1)$, (7)

- All other cases:
  $v[t] = 0$. (8)

We now prove that there exists a graph which realizes the feature vector defined by these rules if and only if M contains a matching.

Lemma 1 *If M has a matching then v is realizable.*

Proof. Suppose M has a matching M'. Define G to be a graph whose vertex set consists of one vertex labeled by z, $|M|$ vertices labeled by y, and one vertex labeled by i for every $i \in A \cup B \cup C$. Put M in one-to-one correspondence with the set of vertices labeled by y, and for each triple $(a_i, b_j, c_k) \in M$, let its corresponding vertex in G be adjacent to the three vertices labeled by a_i, b_j, and c_k; furthermore, if $(a_i, b_j, c_k) \in M'$ then let this vertex also be adjacent to the unique vertex labeled by z. See Fig. 5. It is straightforward to verify that G realizes the feature vector v defined above. □

Lemma 2 *If v is realizable then M has a matching.*

Proof. Suppose there exists a graph G which realizes v. To prove the lemma, we will show that G must have the structure of the graph shown in Fig. 5 and that the q vertices labeled by y

which are adjacent to the vertex labeled by z induce a matching in M. Below, for any $\Sigma' \subseteq \Sigma$, all vertices which are labeled by symbols in Σ' are referred to as Σ'-*vertices*, and vertices which are adjacent to the vertex labeled by z are called *key vertices*.

Firstly, by rules (1), (3), and (7), the subgraph of G induced by $\{y, z\}$-vertices consists of a star graph having a center labeled by z and q non-center $\{y\}$-vertices, along with $|M| - q$ isolated $\{y\}$-vertices. Secondly, by rules (2) and (8), every $(A \cup B \cup C)$-vertex in G is adjacent to $\{y\}$-vertices only; we next consider *how* the $(A \cup B \cup C)$-vertices are attached.

For any pair i, j belonging to exactly one of the three sets A, B, and C, there are no paths of the form (i, y, j) in the feature vector v according to the definition of $s(i, j)$ and rule (6). It follows that each $\{y\}$-vertex is adjacent to at most one A-vertex, at most one B-vertex, and at most one C-vertex. Moreover, if some $\{y\}$-vertex was adjacent to fewer than three $(A \cup B \cup C)$-vertices then there would be less than $3 \cdot |M|$ edges in G between $\{y\}$-vertices and $(A \cup B \cup C)$-vertices, which is impossible because rule (2) implies that the total number of such edges is $\sum_{i \in A \cup B \cup C} s(i) = 3 \cdot |M|$. Therefore, each $\{y\}$-vertex is adjacent to exactly one A-vertex, exactly one B-vertex, and exactly one C-vertex.

Next, let w be any $\{y\}$-vertex and let $a_i \in A$, $b_j \in B$, $c_k \in C$ be the labels of the three $(A \cup B \cup C)$-vertices adjacent to w. Rule (6) ensures that: (i) a_i and b_j must appear together in at least one triple of M, (ii) a_i and c_k must appear together in at least one triple of M, and (iii) b_j and c_k must appear together in at least one triple of M. Then, the pairwise consistency constraint implies that all three of a_i, b_j, c_k appear together in a triple of M, i.e., $(a_i, b_j, c_k) \in M$. We have just proved that every $\{y\}$-vertex in G induces a triple consisting of the labels of its three adjacent $(A \cup B \cup C)$-vertices which indeed belongs to M.

Finally, only $\{y\}$-vertices may be key vertices by rules (3) and (8). Rule (4) implies that two different key vertices can never be adjacent to the same $(A \cup B \cup C)$-vertex, so all triples induced by key vertices must be pairwise disjoint. Thus, G has the structure shown in Fig. 5, and the q triples from M induced by the key vertices constitute a matching of M. □

By Lemmas 1 and 2, we obtain:

Theorem 4 *GIPF is strongly NP-hard even if restricted to $K = 2$.*

It follows from this theorem that there does not exist a pseudo polynomial time algorithm for GIPF unless P=NP. The above theorem remains valid even for the tottering case because every tottering path has the form of (u, v, u) in the case of $K = 2$ and thus does not affect the structure of the proof.

5.2 Hardness of GIPF for trees

We prove that GIPF is strongly NP-hard even for trees and $K = 3$. For that purpose, we provide a pseudo polynomial time reduction from the following problem:

3-PARTITION

Instance: A set $X = \{x_1, x_2, \ldots, x_{3m}\}$ and a positive integer B, where each x_i has integer weight $w(x_i)$.

Question: Is there a partition of X into $A_1, A_2, \ldots, A_m$ such that each A_h consists of three elements and $\sum_{x_j \in A_h} w(x_j) = B$ holds for each A_h ?

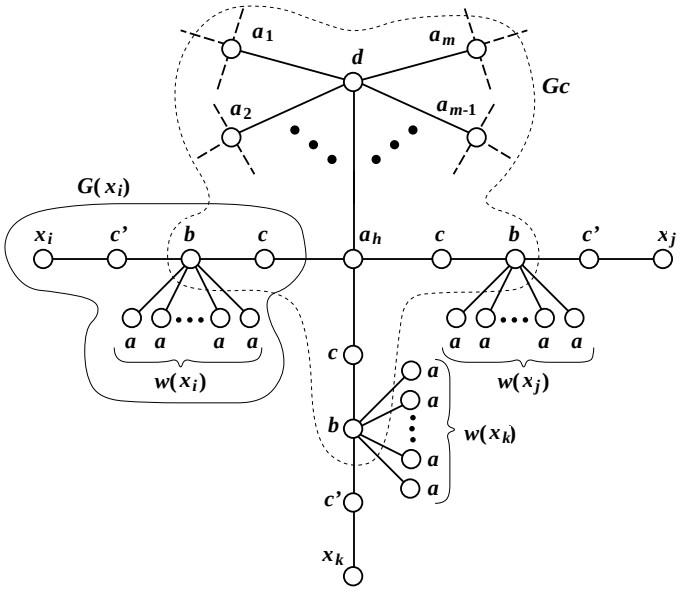


Figure 6: Reduction from 3-PARTITION to GIPF, where a_h corresponds to set $A_h = \{x_i, x_j, x_k\}$.

It is known that 3-PARTITION is strongly NP-hard even when each $w(x_i)$ satisfies $B/4 < w(x_i) < B/2$. This assumption assures that each set consists of exactly three elements. We adopt this assumption below.

We construct a feature vector of level 3 (i.e., $K = 3$), which is to be constructed from subgraphs of the target (tree) graph $G(V, E)$, where $G(V, E)$ corresponds to a solution of 3-PARTITION.

We let

$$\Sigma \;=\; X \;\cup\; \{a_h | h = 1, \ldots, m\} \;\cup\; \{a, b, c, c', d\}.$$

For each x_i, we construct a subtree $G(x_i)$ as shown in Fig. 6. It is to be noted that we identify labels and vertices unless there is confusion.

Each $G(x_i)$ is called a *block*. Note that there are $w(x_i)$ vertices with label a in $G(x_i)$. Note also that three blocks will be connected to the same vertex labeled a_h, though it is not specified by the feature vector which blocks are connected to the same vertex.

Next, we connect the vertex with label d to m vertices with label a_h as in Fig. 6. We call this subgraph the *center star* (denoted by G_c in Fig. 6).

The feature vector $\boldsymbol{v}$ is constructed from the following paths:

- All paths up to length three in all blocks and in the center star,
- For each a_h, we construct B paths of the form of a-b-c-a_h and the corresponding B paths in the reverse direction.

It is not difficult to see that there exists a graph $G(V, E)$ such that $\boldsymbol{f}_K(G) = \boldsymbol{v}$ if and only if there exists a solution to 3-PARTITION, where a set of blocks connecting with a_h correspond to A_h. The necessity is rather obvious because the number of paths a_h-c-b-a is equal to B if and

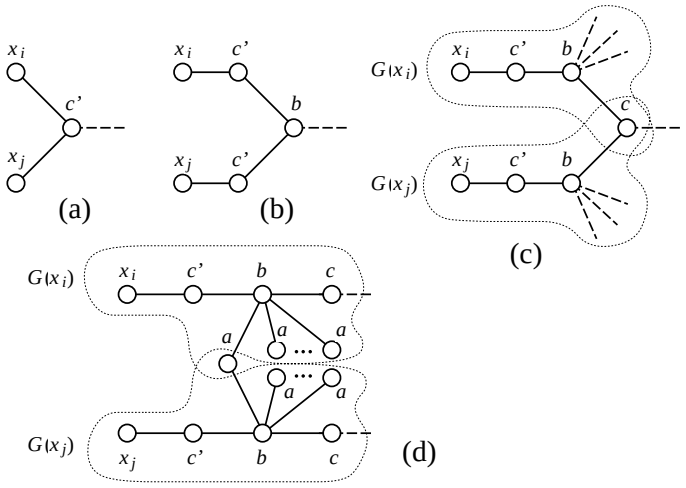


Figure 7: Blocks intersecting with each other.

only if the sum $w(x_i) + w(x_j) + w(x_k)$ is equal to B, where the vertex labeled a_h shares edges between $G(x_i)$, $G(x_j)$, and $G(x_k)$. In order to prove the sufficiency, we need some lemmas.

Lemma 3 *Let H be a graph for which $\boldsymbol{f}_3(H) = \boldsymbol{v}$ holds. Then, H contains the center star G_c and blocks $G(x_i)$ for all x_i.*

Proof. The claim on G_c is almost obvious since each vertex labeled d or a_h is unique in our graph G, and therefore paths d-a_h-c-b are unique in $\boldsymbol{v}$ for each h ($h = 1, \ldots, m$). These paths intersect only at d because there are paths labeled neither a_h-c-$a_{h'}$ nor c-b-c.

The claim on $G(x_i)$ is proven in a similar way by observing that for each x_i, there exists exactly one vertex labeled x_i. □

Lemma 4 *Let H be a graph for which $\boldsymbol{f}_3(H) = \boldsymbol{v}$ holds. Then, for any pair $(x_i, x_j) \in X^2$ ($i \neq j$), $G(x_i)$ and $G(x_j)$ do not intersect with each other (i.e., $V(G(x_i)) \cap V(G(x_j)) = \emptyset$).*

Proof. By Lemma 3, we can assume that there exists a block $G(x_i)$ for each $x_i \in X$.

Suppose that $G(x_i)$ and $G(x_j)$ intersect with each other. Then, they must share a vertex labeled either c', b, c, or a.

Suppose that $G(x_i)$ and $G(x_j)$ share a vertex labeled c'. Then, $\boldsymbol{v} = \boldsymbol{f}_3(G)$ must have a path labeled x_i-c'-x_j (Fig. 7(a)). Since $\boldsymbol{v} = \boldsymbol{f}_3(G)$ contains no paths labeled x_i-c'-x_j, no pair of blocks intersect at c'. Similarly, blocks intersect neither at b, at c, nor at a because $\boldsymbol{v}$ does not contain any path labeled c'-b-c', b-c-b, or b-a-b (Fig. 7(b)(c)(d)).

Hence neither of the above cases happens and thus any blocks can not intersect with each other. □

Since the reduction satisfies the conditions for a pseudo polynomial time transformation [11], we have:

Theorem 5 *GIPF is strongly NP-hard even for trees and $K = 3$.*

We can see that Theorem 5 still holds in the tottering case. In that case, we use the same construction of the feature vector $\boldsymbol{v}$ except that tottering paths are also included.

Since we can discriminate tottering paths from non-tottering paths both starting from x_i or a_h by using information on vertex labels and non-tottering paths beginning from x_i and a_h uniquely determine G_c and $G(x_i)$ respectively, Lemma 3 holds.

In order to prove Lemma 4, we use the same argument as in the non-tottering case. However, there can exist c'-b-c', b-c-b, and b-a-b paths and thus we need to use counting arguments. For example, suppose that the case shown in Fig. 7(b) occurs. From the construction, we can see that $\boldsymbol{v}$ specifies that both the number of edges labeled c'-b and the number of paths labeled c'-b-c' are m. However, if there exists a substructure shown in Fig. 7(b), the number of paths labeled c'-b-c' exceeds m, a contradiction. Therefore, Lemma 4 still holds for the tottering case.

Finally, it is straight-forward to see that the reduction satisfies the conditions for a pseudo polynomial time transformation.

5.3 Hardness of GIPF for trees of bounded degree

In Theorem 5, the maximum degree of graph G and the size of Σ were not bounded. However, we can modify the above reduction for trees of bounded degree 4 and of a fixed Σ. Let $L = \lceil \log \max(B, m) \rceil$. Then we transform blocks and the center star as in Fig. 8, where labels of vertices should be defined appropriately using a fixed Σ. In this transformation, each x_i (resp. a_h) is encoded by using a binary code and each high degree vertex is represented by a binary tree. It should be noted that the maximum degree is 4 in the transformed graph.

In this case, we use a feature vector of level $4L$. Then, it is straightforward to see the correctness of the reduction because paths beginning from the fragment encoding x_i and from the fragment encoding a_h uniquely determines subtrees corresponding to $G(x_i)$ and G_c, respectively. It can also be seen that both the number of dimensions of a feature vector and the size of G are bounded by polynomials in B and m. Moreover, each coordinate value of a feature vector is also bounded by a polynomial in B and m. From these, we can see that the reduction satisfies the conditions of a pseudo polynomial time transformation.

Theorem 6 *GIPF is strongly NP-hard even for trees of bounded degree 4 and of a fixed Σ.*

Theorem 6 again holds in the tottering case. In order prove it, we modify the construction of blocks and the center star as in Fig. 9(A), replace each edge with a path of length 3 as in Fig. 9(B) where α and β are newly introduced symbols, and use a feature vector of level $3 \times 6L$. Replacement of edges enables discrimination of non-tottering paths from tottering paths, by which the structures of $G(x_i)$s and G_c are uniquely determined. The structure of T_i (shown in Fig. 9(A)) was modified so that any node encoding a_h cannot know the number of leaves labeled a in T_i even using tottering paths, although nodes encoding a_h can know the total number of leaves labeled a in T_i, T_j, and T_k (this number must be B). The total number of tottering paths is $O(n \cdot \sum_{k=0}^{18L} 4^k)$, which is still polynomial in B and m. Therefore, the reduction satisfied the condition of a pseudo polynomial time transformation and thus the theorem holds.

6 Conclusion

We have presented polynomial time algorithms for inferring sequences from feature vectors corresponding to the spectrum kernel and for inferring trees of bounded degree from feature

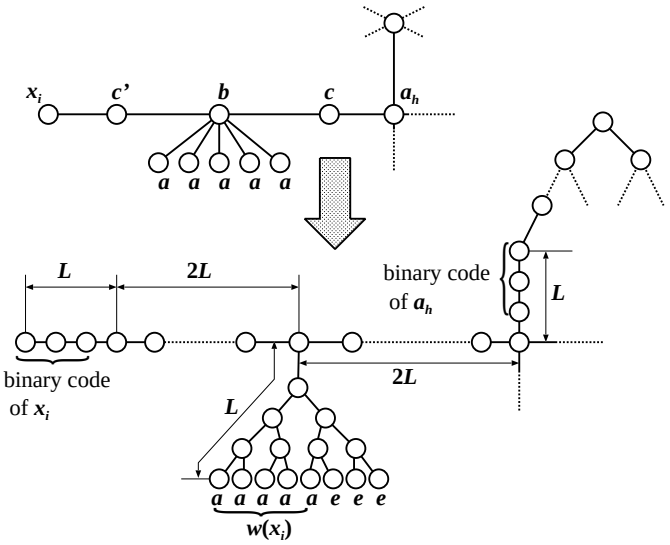


Figure 8: Transformation of blocks and the center star for Theorem 6.

vectors consisting of frequency of paths of fixed length under a fixed alphabet. The results on inference of a graph (GIPF) are summarized as follows, where K is the maximum length of paths, and D is the maximum degree of graphs:

- GIPF can be solved in polynomial time for trees if D, $|\Sigma|$ and K are bounded by constants,
- GIPF is strongly NP-hard for $K = 2$ if D and $|\Sigma|$ are unbounded,
- GIPF is strongly NP-hard for trees and $K = 3$ if D and $|\Sigma|$ are unbounded,
- GIPF is strongly NP-hard for trees if D and $|\Sigma|$ are bounded by constants but K is not bounded.

It is to be noted that GIPF can be solved in polynomial time both for trees and for general graphs if $K = 1$ [21].

From a theoretical viewpoint, there still remain complexity gaps between the positive and negative results. For example, the complexity (polynomial or NP-hard) of the following cases should be studied:

- inference of a tree when all non-tottering paths are given under a fixed alphabet,
- inference of a graph of bounded degree from paths with constant K under a fixed alphabet.

Another interesting future work is to develop approximation algorithms for NP-hard cases.

From a practical viewpoint, the proposed algorithms for trees are not useful because the degrees of polynomials are quite large both in time and space: it is $O(n^6)$ even for $K = 1$ and $\Sigma = \{0, 1\}$. Therefore, faster and/or practical algorithms should be developed. Indeed, after a preliminary version of this paper [1] was published, branch-and-bound algorithms were developed for tree-like structures [2, 10, 14]. However, these algorithms can only handle small or medium-size chemical compounds having tree-like structures. Therefore, further improvements should

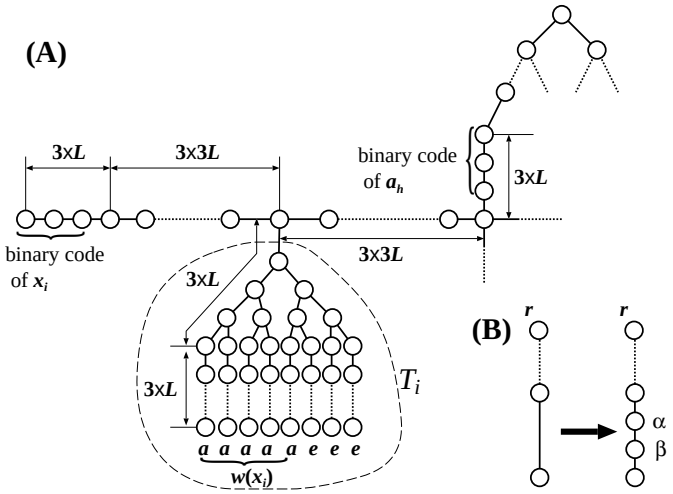


Figure 9: (A) Construction of blocks and the center star for the tottering case of Theorem 6, (B) Replacement of an edge by a path of length 3.

be done. It is to be noted that the tables used in the dynamic programming algorithms in this paper are sparse and thus we may be able to ignore most of the entries in the tables. Therefore, a dynamic programming-based approach might be useful if combined with a branch-and-bound procedure.

In this paper, we considered feature vectors defined by path frequency. However, probabilities of paths are used in the marginalized graphs kernels [15]. Therefore, extensions for such cases should be studied.

Acknowledgements

We thank Hirotaka Ono for useful discussions. The work of TA was partially supported by Grant-in-Aid "Systems Genomics" from MEXT, Japan, and the work of JJ was funded by the Special Coordination Funds for Promoting Science and Technology, Japan.

References

[1] T. Akutsu, D. Fukagawa, Inferring a graph from path frequency, in: Proc. 16th Annual Symposium on Combinatorial Pattern Matching, Lecture Notes in Computer Science, vol. 3537, 2005, pp. 371–382.

[2] T. Akutsu, D. Fukagawa, Inferring a chemical structure from a feature vector based on frequency of labeled paths and small fragments, in: Proc. 5th Asia-Pacific Bioinformatics Conference, Imperial College Press, London, 2007, pp. 165–174.

[3] T. Asano, An $O(n \log \log n)$ time algorithm for constructing a graph of maximum connectivity with prescribed degrees, J. Comput. System Sci. 51 (1995) 503–510.

[4] G.H. Bakir, J. Weston, B. Schölkopf, Learning to find pre-images, Advances in Neural Information Processing Systems, 16 (2004) 449–456.

[5] G.H. Bakir, A. Zien, K. Tsuda, Learning to find graph pre-images, in: Proc. the 26th DAGM symposium. Lecture Notes in Computer Science, vol. 3175, 2004, pp. 253–261.

[6] C. Cortes, V. Vapnik, Support vector networks, Machine Learning 20 (1995) 273–297.

[7] C. Cortes, M. Mohri, J. Weston, A general regression technique for learning transductions, in: Proc. 22nd International Conference on Machine Learning, 2005, pp. 153–160.

[8] C. Cortes, M. Mohri, J. Weston, A general regression framework for learning string-to-string mappings, in: Bakir et al. (Eds.), Predicting Structured Data, The MIT Press, 2007, pp. 143–168.

[9] Y. Dinitz, A. Itai, M. Rodeh, On an algorithm of Zemlyachenko for subtree isomorphism. Inform. Proc. Lett. 70 (1999) 141–146.

[10] H. Fujiwara, J. Wang, L. Zhao, H. Nagamochi, T. Akutsu, Enumerating tree-like chemical graphs with given path frequency, J. Chem. Inf. Model. 48 (2008) 1345–1357.

[11] M.R. Garey, D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., New York, 1979.

[12] T. Gärtner, P. A. Flach, S. Wrobel, On graph kernels: Hardness results and efficient alternatives, in: Proc. 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, 2003, pp. 129–143.

[13] Y. Hizukuri, Y. Yamanishi, O. Nakamura, F. Yagi, S, Goto, M. Kanehisa, Extraction of leukemia specific glycan motifs in humans by computational glycomics, Carbohydrate Research, 340 (2005) 2270–2278.

[14] Y. Ishida, Y. Kato, L. Zhao, H. Nagamochi, T. Akutsu, Branch-and-bound algorithms for enumerating treelike chemical graphs with given path frequency using detachment-cut, J. Chem. Inf. Model. 50 (2010) 934–946.

[15] H. Kashima, K. Tsuda, A. Inokuchi, Marginalized kernels between labeled graphs, in: Proc. 20th International Conference on Machine Learning, 2003, pp. 321–328.

[16] J. Lauri, R. Scapellato, Topics in Graph Automorphisms and Reconstruction, Cambridge University Press, London, UK, 2003.

[17] C. Leslie, E. Eskin, W.S. Noble, The spectrum kernel: a string kernel for SVM protein classification, in: Proc. Pacific Symposium on Biocomputing 2002, 2002, pp. 564–575.

[18] P. Mahé, N. Ueda, T. Akutsu, J-L. Perret, J-P. Vert, Extensions of marginalized graph kernels, in: Proc. 21st International Conference on Machine Learning, 2004, pp. 552–559.

[19] O. Maruyama, S. Miyano, Inferring a tree from walks, Theoretical Computer Science, 161 (1996) 289–300.

[20] R.B. Nachbar, Molecular evolution: automated manipulation of hierarchical chemical topology and its application to average molecular structures, Genetic Programming and Evolvable Machines 1 (2000), 57–94.

[21] H. Nagamochi, A detachment algorithm for inferring a graph from path frequency, Algorithmica 53 (2009) 207–224.

[22] P.A. Pevzner, Computational Molecular Biology. An Algorithmic Approach, The MIT Press, MA, 2000.

[23] V. Raghavan, Bounded degree graph inference from walks, J. Comput. System Sci. 49 (1994) 108–132.

[24] B. Schölkopf, K. Tsuda, J-P. Vert (Eds.), Kernel Methods in Computational Biology, The MIT Press, MA, 2004.

[25] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge Univ. Press, Cambridge, UK, 2004.

[26] N. Shervashidze, K. Borgwardt, Fast subtree kernels on graphs, Advances in Neural Information Processing Systems 22 (2009) 1660–1668.

[27] L. Song, X. Zhang, A. Smola, A. Gretton, B. Schölkpf, Tailoring density estimation via reproducing kernel moment matching, in: Proc. 25th International Conference on Machine Learning, 2008, pp. 992–999.

[28] H.M. Vinkers, M.R. de Jonge, F.F.D Daeyaert, J. Heeres, L.M.H. Koymans, J.H. van Lenthe, P.J. Lewi, H. Timmerman, K. van Aken, P.A.J. Janssen, Synopsis: synthesize and optimize system in silico, J. Med. Chem. 46 (1993) 2765–2773.

[29] X. Zhang, L. Song, A. Gretton, A.J. Smola, Kernel measures of independence for non-iid data, Advances in Neural Information Processing Systems 21 (2008) 1937–1944.