Journal of Information Processing Vol.23 No.2 210–220 (Mar. 2015)

[DOI: 10.2197/ipsjjip.23.210]

Regular Paper

Indexing Method for Hierarchical Graphs based on Relation among Interlacing Sequences of Eigenvalues

Kaoru Katayama[1,a)] Ernest Weke Maina[b)]

Received: March 30, 2014, Accepted: November 10, 2014

Abstract: We propose an indexing method for hierarchical graphs using their eigenvalues in order to detect those that are substructures or superstructures of a hierarchical graph given as a query efficiently. The index construction and the query processing are based on a relation among three interlacing sequences of eigenvalues of hierarchical graphs. We also propose a matrix representation for a hierarchical graph. Hierarchical graphs are decomposed to improve the filtering effect of the index and reduce the computational cost of both the index construction and the query processing. We evaluate the effectiveness of the proposed method by experiments.

Keywords: index, hierarchical graph, interlace, eigenvalue

1. Introduction

Hierarchical structures appear in various real-world data on their own, or as a result of analyzing them. Molecular structures and social networks naturally have hierarchies. Clustering a set of data leads to a hierarchical organization of their subsets. Such data also have various relations among them besides hierarchies and they are often represented as graphs. So there are many works on managing and mining graphs [1]. A graph with a hierarchical structure, a *hierarchical graph* is a natural extension of a graph. Since it has more expressive power than a graph as a data model, it is used to represent data with a complex structure more precisely. As with searching subgraphs or supergraphs of a graph in graph-based applications, searching substructures or superstructures of a hierarchical graph is a basic task in applications dealing with complex data. For example, in a meta-search engine, WhatsOnWeb [2], a query is expressed as a hierarchical graph. Barchall et al. [3] use hierarchical graphs as abstract representations of chemical structures. Their substructures are searched according to the hierarchies.

In this paper, we propose an indexing method for hierarchical graphs to detect those that are superstructures of a hierarchical graph given as a query efficiently. We represent hierarchical graphs as matrices and construct the index based on a relation of their eigenvalues. Given a hierarchical graph as a query, we filter hierarchical graphs which are not answers to the query and find candidates for the answers efficiently by comparing eigenvalues of hierarchical graphs and the query. The same approach can be used for constructing an index to detect substructures of a query. The key fact in our indexing method is a relation among three sequences α, β and γ of eigenvalues of hierarchical graphs, that is, if γ interlaces β and β interlaces α, then γ interlaces α. The word "interlace" comes from the interlace theorem [4]. We give the definition of this word and a simple proof of the above fact later. The proposed index is based on the above fact and the interlace theorem. We also propose a symmetric matrix representation for a hierarchical graph where a substructure of the hierarchical graph corresponds to the principal submatrix in order to satisfy the assumption of the theorem. Commonly used matrix representations such as adjacency matrices do not have this property. In order to improve the filtering effect of the index and reduce the computation cost of the index construction and the query processing, hierarchical graphs are decomposed according to labels of the vertices and the edges. For graphs, various indexing methods are proposed [5]. On the other hand, there are few works on indexing hierarchical graphs for detecting their superstructures or substructures to the authors' knowledge. We keep our proposition as simple as possible and show the basic performance of our approach by experiments. The performance of the proposed index can be improved by combining with the various existing techniques for graphs. We think that this approach is more informative for understanding the advantages and disadvantages of our proposition and improving it in future. Since graphs are hierarchical graphs without hierarchy, our approach is applicable to them.

The rest of the paper is organized as follows. We discuss related work in Section 2. Definitions on graphs, their matrix representation and the interlace theorem on eigenvalues of symmetric matrices are presented in Section 3. Hierarchical graphs and their matrix representation are defined in Section 4. In Section 5, we explain the basic idea of our indexing method. The detailed algorithms for constructing the index structure and processing a query are shown in Section 6. In Section 7, we show the efficiency and scalability of our index by experiments. We conclude in Section 8.

[1] Graduate School of System Design, Tokyo Metropolitan University, Hino, Tokyo 191–0065, Japan
[a)] kaoru@tmu.ac.jp
[b)] ewmaina@gmail.com

2. Related Work

Hierarchical graphs are used as data models in various applications such as content-based image retrieval [6], query expressions for a web search engine [2] and representations of chemical structures [3]. Although there exists many works on indexing (not hierarchical) graphs [5], there are few works on indexing hierarchical graphs for retrieving their substructures or superstructures to the authors' knowledge. Shokoufandeh et al. [6] propose an indexing method for hierarchical image features. The hierarchical structure of the features is represented as an unlabeled directed acyclic graph. The features are embedded into low-dimensional vector space using eigenvalues of a matrix representation composed of adjacency matrices of subgraphs of the graph. Answers to a query are retrieved using nearest-neighbor search. Their indexing method is for directed acyclic graphs and not for hierarchical graphs which are our concern in this paper. Demirci et al. [7] take a similar approach to Shokoufandeh et al. [6] for indexing image features. They represent image features as unlabeled undirected graphs and encode them as eigenvalues of their laplacian matrices. The features are embedded into vector space by the eigenvalues and answers to a query are retrieved using nearest-neighbor search.

Zhang et al. [8] propose an indexing method using eigenvalues of graphs for XML documents. They represent a XML document and a XPath query as labeled directed acyclic graphs and use the maximum eigenvalues and the minimum ones of anti-symmetric matrices representing the graphs as keys of the index which is a B^+ tree. In processing a query, they perform "range query" based on the "eigenvalue containment property" [8] in the B^+ tree, which is a relation among the maximum eigenvalues and the minimum ones of two anti-symmetric matrices representing a directed acyclic graph and its induced subgraph. On the other hand, our index structure is the tree which reflects directly the interlacing relation between eigenvalues of a (hierarchical) graph and its (hierarchical) subgraph. Therefore it is not necessary for us to perform "range query" in the index. In addition, our indexing method can be used for detecting not only induced subgraphs of labeled directed acyclic graphs but also general (hierarchical) subgraphs of (hierarchical) graphs. Zou et al. [9] also propose an indexing method, GCoding-tree, using some eigenvalues of a graph as its feature. The structure of GCoding-tree is based on combinatorial structures of each graph in a graph database. Eigenvalues of graphs are used as encoding for the graphs. They are stored in nodes of GCoding-tree to filter graphs which are not answers to a query. Shokoufandeh et al. [10] propose another indexing method for object recognition. It maps the tree representing features of an object into a vector space by eigenvalues of the adjacency matrix of the tree.

Most of graph indicies take combinatorial approaches in index construction and query processing. Fg-index [11] is such a graph index using only combinatorial methods. It is constructed from frequent subgraphs in a graph database. In order to find such subgraphs, a graph mining algorithm, gSpan [12] is used. Cheng et al. [13] have improved Fg-index and proposed a new indexing method called Fg*-index. They have addressed the problem of the number of frequent subgraphs generated by a graph mining algorithm and the size of Fg-index. GraphGrep [14] and Closure-tree [15] are the indexes for finding supergraphs of a query. GraphGrep uses all paths from each vertex of each graph in a database for index construction. Closure-tree is constructed hierarchically from generalized graphs which summarizes its descendant graphs. Since a hierarchical graph has a more complex structure than a graph, it costs much more to find frequent hierarchical subgraphs or all paths from each vertex of a hierarchical graph like graphs. We avoid such costly methods.

3. Preliminaries

We study hierarchical graphs composed of a *labeled undirected simple graph* $g = (V^g, E^g, L(V)^g, L(E)^g, \mu^g, \nu^g)$. V^g or $V(g)$ is a set of vertices. E^g is a set of edges where an edge e in E^g is an ordered pair (v_1, v_2) of vertices v_1 and v_2 in V^g. $L(V)^g$ and $L(E)^g$ are sets of labels of vertices and edges, respectively. μ^g and ν^g are onto-mappings $V^g \to L(V)^g$ and $E^g \to L(E)^g$, respectively. If, for any $(v_1, v_2) \in E^g$, there is the edge $(v_2, v_1) \in E^g$, g is called an *undirected* simple graph. We simply call a labeled undirected simple graph a graph. For a vertex v of g, if there is not an edge (v, v') or $(v', v) \in E^g$, we call v an *isolated vertex*. In this paper, labels of vertices and edges are positive numbers. If they are not given as positive numbers, we change each label to a positive number in a suitable way for applications. For succinct description, we denote each element of a graph g as a symbol followed by a superscript to specify g as above. For example, we denote a set of vertices of a graph g as V^g. Examples of the following definitions are included in ones of hierarchical graphs in the next section.

Definition 1 (Subgraph) A graph $g_1 = (V^1, E^1, L(V)^1, L(E)^1, \mu^1, \nu^1)$ is a *subgraph* of another graph $g_2 = (V^2, E^2, L(V)^2, L(E)^2, \mu^2, \nu^2)$, if there is an injection $i : V^1 \to V^2$ which satisfies the following conditions for any $(v_1, v_2) \in E^1$.

- $(i(v_1), i(v_2)) \in E^2$
- $\mu^1(v_1) = \mu^2(i(v_1))$ and $\mu^1(v_2) = \mu^2(i(v_2))$
- $\nu^1(v_1, v_2) = \nu^2(i(v_1), i(v_2))$

The injection i is called *subgraph isomorphism*.

3.1 Eigenvalues and Subgraphs

For eigenvalues of a real symmetric matrix and its submatrix, there is the following relation known as the interlace theorem [4]. ${}^tS$ is the transpose of a matrix S.

Definition 2 Let $\{\alpha_i\}_{i=1,\ldots,n}$ and $\{\beta_j\}_{j=1,\ldots,m}$ be two ordered sequences of real numbers where $m < n$, $\alpha_1 \le \alpha_2 \le \cdots \le \alpha_n$ and $\beta_1 \le \beta_2 \le \ldots \le \beta_m$, respectively. We say that $\{\beta_j\}_{j=1,\ldots,m}$ *interlaces* $\{\alpha_i\}_{i=1,\ldots,n}$, if the following condition is satisfied for $k = 1, \ldots, m$.

$$\alpha_k \le \beta_k \le \alpha_{k+(n-m)}$$

Theorem 1 (Interlace Theorem) Given a real $n \times m$ matrix S such that ${}^tSS = I$ and a symmetric $n \times n$ matrix A, the eigenvalues of a $m \times m$ matrix $B = {}^tSAS$ interlace those of A.

When a graph g_1 and its subgraph g_2 are represented as the matrices M^1 and M^2 which satisfy the assumption of this theorem, respectively, the eigenvalues of M^2 interlace the eigenvalues of

M^1. If the eigenvalues of M^2 do not interlace the eigenvalues of M^1, g_2 is not a subgraph of g_1 by the contraposition.

3.2 Matrix Representation of Graphs

In defining a matrix representation of a hierarchical graph later, we use the following *extended incidence matrix* for a graph [16].

Definition 3 A *labeled incidence matrix* $N^g = (n^g_{ij})$ of a *undirected* graph $g = (V^g, E^g, L(V)^g, L(E)^g, \mu^g, \nu^g)$ is the $|V^g| \times |E^g|$ matrix as follows.

$$n^g_{ij} := \begin{cases} \nu^g(e_j) & \text{if } e_j \in E^g \text{ is incident to } v_i \in V^g \text{ for } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

When $\nu^g(e)$ is 1 for any edge $e \in E^g$, N^g is an ordinary *incidence matrix* of a graph. Depending on applications, we may define this matrix by assigning labels of vertices to the elements instead of labels of edges.

Definition 4 An *extended incidence matrix* C^g of a directed graph $g = (V^g, E^g, L(V)^g, L(E)^g, \mu^g, \nu^g)$ is the matrix

$$\begin{bmatrix} P & N^g \\ {}^tN^g & Q \end{bmatrix}$$

where $N^g = (n_{ij})$ is the $|V^g| \times |E^g|$ labeled incidence matrix of g, $P = (p_{ij})$ and $Q = (q_{ij})$ are $|V^g| \times |V^g|$ and $|E^g| \times |E^g|$ matrices as follows, respectively.

$$p_{ij} := \begin{cases} \mu^g(v_i) & \text{if } i = j \text{ for } v_i \in V^g, \\ 0 & \text{otherwise.} \end{cases}$$

$$q_{ij} := \begin{cases} \nu^g(e_i) & \text{if } i = j \text{ for } e_i \in E^g, \\ 0 & \text{otherwise.} \end{cases}$$

3.3 Decomposing Graphs by Labels

We use the following decomposition of graphs according to labels assigned to vertices and edges of graphs in defining the decomposition of hierarchical graphs [16].

Definition 5 For a graph g, the graph $g[\mu^g(v)]$ decomposed by the label $\mu^g(v)$ of a vertex $v \in V^g$ is as follows.

- $V^{g[\mu^g(v)]} = \{v' \in V^g | \mu^g(v') = \mu^g(v)\}$
- $E^{g[\mu^g(v)]} = \{(v_1, v_2) \in E^g | \mu^g(v_1) = \mu^g(v_2) = \mu^g(v)\}$
- $L(V)^{g[\mu^g(v)]} = \{\mu^g(v)\}$
- $L(E)^{g[\mu^g(v)]} = \{\nu^g(e) | e \in E^{g[\mu^g(v)]}\}$
- $\mu^{g[\mu^g(v)]}$ is a restriction of μ^g to $V^{g[\mu^g(v)]}$
- $\nu^{g[\mu^g(v)]}$ is a restriction of ν^g to $E^{g[\mu^g(v)]}$

The graph $g[\nu^g(e)]$ decomposed by the label $\nu^g(e)$ of an edge $e \in E^g$ is as follows.

- $E^{g[\nu^g(e)]} = \{e' \in E^g | \nu^g(e') = \nu^g(e)\}$,
- $V^{g[\nu^g(e)]} = \{v_1, v_2 \in V^g | (v_1, v_2) \in E^{g[\nu^g(e)]}\}$,
- $L(E)^{g[\nu^g(e)]} = \{\nu^g(e)\}$,
- $L(V)^{g[\nu^g(e)]} = \{\mu^g(v) | v \in V^{g[\nu(e)]}\}$,
- $\mu^{g[\nu^g(e)]}$ is a restriction of μ^g to $V^{g[\mu^g(v)]}$
- $\nu^{g[\nu^g(e)]}$ is a restriction of ν^g to $E^{g[\mu^g(v)]}$

The graph $g[\mu^g(v)][\nu^g(e)]$ means $(g[\mu^g(v)])[\nu^g(e)]$, that is, the graph is decomposed by the label $\mu^g(v)$ of a vertex v and then decomposed by the label $\nu^g(e)$ of an edge e.

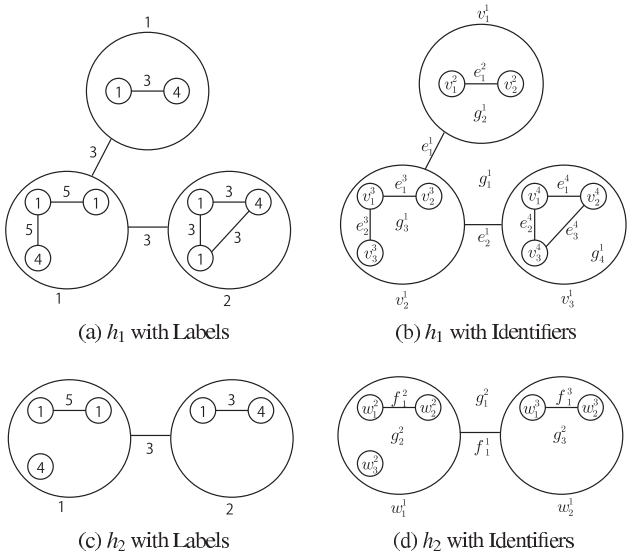

Fig. 1 Hierarchical graph h_1 and its hierarchical subgraph h_2.

4. Hierarchical Graph and Matrix Representation

We define a hierarchical graph and a hierarchical subgraph as an extension of concept of graph and subgraph. Although there are slightly different types of hierarchical graphs, we focus on a simple one shown in **Fig. 1**. A vertex of a graph of which a hierarchical graph consists contains another graph. Since a graph is a special hierarchical graph without hierarchy, we can also use the proposed methods for graphs.

Definition 6 (Hierarchical Graph) A hierarchical graph h is a tuple (G^h, ϕ^h) where

- G^h is a set of graphs $\{g^h_i | i \in \{1, \ldots, n\}\}$ which contains the root graph r^h,
  The root graph is the uppermost graph in the hierarchy and is not contained in a vertex of another graph in G^h. It may be disconnected as other graphs.
- ϕ^h is a mapping from $\bigcup_{i=1}^n V^{g^h_i}$ to $G^h \setminus \{r^h\}$ where $(\phi^h)^{-1}$ is an injection.
  ϕ^h represents the hierarchical structure of h. For a vertex v of a graph in G^h, $\phi^h(v)$ is a graph contained in v. Every graph in $G^h \setminus \{r^h\}$ is contained in different vertices of graphs in G^h.

We call a substructure and a superstructure of a hierarchical graph a *hierarchical subgraph* and a *hierarchical supergraph*, respectively. That is, a hierarchical subgraph is contained in a hierarchical graph, and a hierarchical supergraph contains a hierarchical graph.

Definition 7 (Hierarchical Subgraph) A hierarchical graph $h_1 = (G^1, \phi^1)$ is a *hierarchical subgraph* of another hierarchical graph $h_2 = (G^2, \phi^2)$, if there is an injection i^h from G^1 to G^2 as follows.

- for any $g \in G^1$, g is a subgraph of $i^h(g) \in G^2$
- for any vertex $v \in V^g$ of any graph $g \in G^1$, if there is a graph $\phi^1(v)$, $\phi^1(v)$ is a subgraph of $\phi^2(i^g(v))$ where $i^g : V^g \to V^{i^h(g)}$ is the subgraph isomorphism.

Example 1 Figure 1 shows examples of a hierarchical graph h_1 and its hierarchical subgraph h_2. h_1 is a hierarchical super-

graph of h_2. The numbers in (a) and (c) are labels of vertices and edges, and the symbols in (b) and (d) are their identifiers. h_1 is (G^1, ϕ^1) where $G^1 = \{g_1^1, g_2^1, g_3^1, g_4^1\}$ and g_1^1 is the root graph of h_1. ϕ^1 is the mapping where $\phi^1(v_1^1) = g_2^1$, $\phi^1(v_2^1) = g_3^1$ and $\phi^1(v_3^1) = g_4^1$. In the same way, h_2 is (G^2, ϕ^2) where $G^2 = \{g_1^2, g_2^2, g_3^2\}$ and g_1^2 is the root graph of h_2. ϕ^2 is the mapping where $\phi^2(w_1^1) = g_2^2$ and $\phi^2(w_2^1) = g_3^2$. h_2 is a hierarchical subgraph of h_1 according to the injection i^h from G^2 to G^1 where $i^h(g_1^2) = g_1^1$, $i^h(g_2^2) = g_3^1$ and $i^h(g_3^2) = g_4^1$. For the subgraph isomorphism $i^g : V^{g_1^2} \to V^{g_1^1}$ between $g_1^2 \in G^2$ and $g_1^1 \in G^1$, $\phi^2(w_1^1) = g_2^2$ and $\phi^2(w_2^1) = g_3^2$ are subgraphs of $\phi^1(i^g(w_1^1)) = g_3^1$ and $\phi^1(i^g(w_2^1)) = g_4^1$, respectively.

We define a matrix representation for a hierarchical graph as follows to have the prerequisites of the interlace theorem. That is, if h^s is a hierarchical subgraph of another hierarchical graph h, there is a matrix S such that $D^{h^s} = {}^tSD^hS$ and ${}^tSS = I$ for the matrices D^h and D^{h^s} representing h and h^s.

Definition 8 An *hierarchical incidence matrix* D^h of a hierarchical graph $h = (G^h, \phi^h)$ where $G^h = \{g_i^h | i = 1, \ldots, n\}$ is the matrix as follows.

$$\begin{bmatrix} C^{g_1^h} & R^{g_1^h, g_2^h} & \cdots & R^{g_1^h, g_n^h} \\ {}^tR^{g_1^h, g_2^h} & C^{g_2^h} & \cdots & R^{g_2^h, g_n^h} \\ \vdots & \vdots & \ddots & \vdots \\ {}^tR^{g_1^h, g_n^h} & {}^tR^{g_2^h, g_n^h} & \cdots & C^{g_n^h} \end{bmatrix}$$

$C^{g_i^h}$ is an extended incidence matrix of g_i^h. $R^{g_i^h, g_j^h} = (r_{kl})$ is the $(|V^{g_i^h}| + |E^{g_i^h}|) \times (|V^{g_j^h}| + |E^{g_j^h}|)$ matrix as follows.

- If a vertex v_k of g_i^h contains g_j^h, each element of the k-th row corresponding to v_k is $\mu^{g_i^h}(v_k)$ and the other elements are 0.
- If a vertex v_l of g_j^h contains g_i^h inversely, each element of the l-th column corresponding to v_l is $\mu^{g_j^h}(v_l)$ and the other elements are 0.
- If g_i^h and g_j^h do not have the above relations, all the elements are 0.

We use labels of vertices as elements of $R^{g_i^h, g_j^h}$ in this definition. Alternatively, labels of edges or other suitable numbers can also be used as the elements depending on applications.

Example 2 For graphs g_1^1, g_2^1 and g_3^1 in hierarchical graph h_1 of Fig. 1, $C^{g_1^1} = \begin{bmatrix} 1&0&0&3&0 \\ 0&1&0&3&3 \\ 0&0&2&0&3 \\ 3&3&0&3&0 \\ 0&3&3&0&3 \end{bmatrix}$, $C^{g_2^1} = \begin{bmatrix} 1&0&3 \\ 0&4&3 \\ 3&3&3 \end{bmatrix}$ and $C^{g_3^1} = \begin{bmatrix} 1&0&0&5&0 \\ 0&1&0&5&5 \\ 0&0&4&0&5 \\ 5&5&0&5&0 \\ 0&5&5&0&5 \end{bmatrix}$, respectively. Since $v_1^1 \in V^{g_1^1}$ of g_1^1 contains g_2^1, $R^{g_2^1, g_1^1}$ is $\begin{bmatrix} 1&0&0&0&0 \\ 1&0&0&0&0 \\ 1&0&0&0&0 \end{bmatrix}$ where 1 is $\mu^{g_1^1}(v_1^1)$ of v_1^1. There is no inclusion relation between g_2^1 and g_3^1. Therefore $R^{g_2^1, g_3^1}$ is $\begin{bmatrix} 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{bmatrix}$. Since $v_2^1 \in V^{g_1^1}$ of g_1^1 contains g_3^1, $R^{g_1^1, g_3^1}$ is $\begin{bmatrix} 0&0&0&0&0 \\ 1&1&1&1&1 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \\ 0&0&0&0&0 \end{bmatrix}$ where 1 is $\mu^{g_1^1}(v_2^1)$ of v_2^1. The hierarchical incidence matrices D^1 and D^2 of h_1 and its hierarchical subgraph h_2 are as follows. For D^1 and D^2, there is the matrix S such that $D^2 = {}^tSD^1S$ and ${}^tSS = I$.

$$D^1 = \begin{bmatrix} C^{g_2^1} & R^{g_2^1, g_1^1} & R^{g_2^1, g_3^1} & R^{g_2^1, g_4^1} \\ {}^tR^{g_2^1, g_1^1} & C^{g_1^1} & R^{g_1^1, g_3^1} & R^{g_1^1, g_4^1} \\ {}^tR^{g_2^1, g_3^1} & {}^tR^{g_1^1, g_3^1} & C^{g_3^1} & R^{g_3^1, g_4^1} \\ {}^tR^{g_2^1, g_4^1} & {}^tR^{g_1^1, g_4^1} & {}^tR^{g_3^1, g_4^1} & C^{g_4^1} \end{bmatrix}$$

$$= \begin{bmatrix} 1&0&3&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&4&3&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 3&3&3&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&1&1&1&0&0&3&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&3&3&1&1&1&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&2&0&3&0&0&0&0&0&2&2&2&2&2&2&2 \\ 0&0&0&3&3&0&3&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&3&3&0&3&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&1&0&0&5&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&1&0&5&5&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&4&0&5&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&5&5&0&5&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&5&5&0&5&0&0&0&0&0&0&0 \\ 0&0&0&0&0&2&0&0&0&0&0&0&0&1&0&3&3&0&3&3 \\ 0&0&0&0&0&2&0&0&0&0&0&0&0&0&1&0&3&3&0&0 \\ 0&0&0&0&0&2&0&0&0&0&0&0&0&0&0&4&0&3&3&3 \\ 0&0&0&0&0&2&0&0&0&0&0&0&0&3&3&0&3&0&0&0 \\ 0&0&0&0&0&2&0&0&0&0&0&0&0&0&3&3&0&3&0&0 \\ 0&0&0&0&0&2&0&0&0&0&0&0&0&3&0&3&0&0&0&3 \end{bmatrix}$$

$$D^2 = \begin{bmatrix} C^{g_1^2} & R^{g_1^2, g_2^2} & R^{g_1^2, g_3^2} \\ {}^tR^{g_1^2, g_2^2} & C^{g_2^2} & R^{g_2^2, g_3^2} \\ {}^tR^{g_1^2, g_3^2} & {}^tR^{g_2^2, g_3^2} & C^{g_3^2} \end{bmatrix} = \begin{bmatrix} 1&0&3&1&1&1&1&0&0&0 \\ 0&2&3&0&0&0&0&2&2&2 \\ 3&3&3&0&0&0&0&0&0&0 \\ 1&0&0&1&0&0&5&0&0&0 \\ 1&0&0&0&1&0&5&0&0&0 \\ 1&0&0&0&0&4&0&0&0&0 \\ 1&0&0&5&5&0&5&0&0&0 \\ 0&2&0&0&0&0&0&1&0&3 \\ 0&2&0&0&0&0&0&0&4&3 \\ 0&2&0&0&0&0&0&3&3&3 \end{bmatrix}$$

$$S = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1 \end{bmatrix}$$

5. Structure of Proposed Index

We explain our approach for constructing the proposed index and processing queries to detect hierarchical supergraphs of a hierarchical graph given as a query. We can filter hierarchical graphs which do not contain the query from a hierarchical graph database, and find candidates for the answers with the index. We call such hierarchical graphs which are *not* hierarchical supergraphs of a query *non hierarchical supergraphs*. The same approach can be used for detecting hierarchical subgraphs of a query. We discuss this briefly later in Section 5.3.

5.1 Detecting Non Hierarchical Supergraphs by Eigenvalues

We decide whether a hierarchical graph h in a database is not a hierarchical supergraph of a query q by comparing the eigenvalues of their hierarchical incidence matrices D^h and D^q defined in the previous section. That is, if eigenvalues of D^q does not interlace eigenvalues of D^h, h is not a hierarchical supergraph of q as is the case of graphs in Section 3.1. We denote a set of eigenvalues of the hierarchical incidence matrix of a hierarchical graph h as $Eig(h)$. We construct a tree-structured index for hierarchical graphs based on the following relation among sequences of real numbers. This is a key fact of our approach.

Proposition 1 Let α, β and γ be three sequences of real numbers. If γ interlaces β and β interlaces α, then γ interlaces α.

Proof Let three sequences of real numbers α, β and γ be $\{\alpha_1, \alpha_2, \ldots, \alpha_l\}$, $\{\beta_1, \beta_2, \ldots, \beta_m\}$ and $\{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ where $\alpha_1 \le \alpha_2 \le \cdots \le \alpha_l$, $\beta_1 \le \beta_2 \le \cdots \le \beta_m$, $\gamma_1 \le \gamma_2 \le \cdots \le \gamma_n$ and $l > m > n$. It is clear that $\alpha_i \le \gamma_i$ for $i = 1, 2, \ldots, n$ from the assumption. We can also show $\gamma_i \le \alpha_{i+(l-n)}$ for $i = 1, 2, \ldots, n$ as follows.

$$\gamma_i \le \beta_{i+(m-n)} \le \alpha_{i+(m-n)+(l-m)} = \alpha_{i+(l-n)} \qquad \square$$

Each node of the tree-structured index contains an identifier of a hierarchical graph and its eigenvalues. When eigenvalues $Eig(h_1)$ of a hierarchical graph h_1 interlace eigenvalues $Eig(h_2)$ of another hierarchical graph h_2, the node which contains $Eig(h_1)$ is a child of the node which contains $Eig(h_2)$. The query processing starts from the root node of the index constructed based on the above manner. We check whether eigenvalues $Eig(q)$ of q interlace eigenvalues in a node of the index. If $Eig(q)$ do not interlace the eigenvalues in the node, we know that not only the hierarchical graph in the node but also all hierarchical graphs in the descendant nodes are not hierarchical supergraphs of q because of the following corollary.

Corollary 1 Let α, β and γ be three sequences of real numbers. If β interlace α and γ does not interlace α, then γ does not interlace β.

Proof By the way of contradiction, we assume that γ interlace β. In this case γ also interlace α by Proposition 1. This is contradiction. □

Here, α, β and γ correspond to sequences of eigenvalues of the following three hierarchical graphs, that is, a hierarchical graph in a node *nod* of the index, a hierarchical graph in a child node of *nod* and a query hierarchical graph, respectively. We describe the detail of algorithms for constructing the index and processing a query in the next section.

5.2 Decomposing Hierarchical Graphs by Labels

In order to reduce the cost of computing eigenvalues of hierarchical graphs, we decompose each hierarchical graph in a database and a query according to labels of vertices and edges. Although there are various ways to decompose hierarchical graphs according to labels, we consider the following ways for simplicity. It is an extension of the decomposition of graphs, which is described in Definition 5. The decomposition of hierarchical graphs also results in a significant improvement in the filtering performance of the proposed index as shown in the experimental evaluation.

Definition 9 For a hierarchical graph $h = (G^h, \phi^h)$ where $G^h = \{g_i^h | i = 1, \ldots, m\}$, the hierarchical graph $h[\mu^g(v)] = (G^{h[\mu^g(v)]}, \phi^{h[\mu^g(v)]})$ decomposed by the label $\mu^g(v)$ of a vertex v of a graph g in G^h is as follows.

- $\phi^{h[\mu^g(v)]}$ is a restriction of ϕ^h to $\{v' \in \bigcup_{i=1}^m V(g_i^h) | \mu^g(v') = \mu^g(v)\}$.
- $G^{h[\mu^g(v)]}$ is a set of graphs $\{g_i^h[\mu^g(v)] | i = 1, \ldots, m\}$ where the root graph is the union of $r^h[\mu^g(v)]$ and a set of graphs $\{g_i^h[\mu^g(v)] | \phi^{h[\mu^g(v)]}(v') \neq g_i^h[\mu^g(v)]$ for any $v' \in \bigcup_{i=1}^m V(g_i^h[\mu^g(v)])\}$. That is, $g_i^h[\mu^g(v)]$ where the vertex containing it is deleted becomes a part of the root graph.

The hierarchical graph $h[\nu^g(e)] = (G^{h[\nu^g(e)]}, \phi^{h[\nu^g(e)]})$ decomposed by the label $\nu^g(e)$ of an edge e in $\bigcup_{i=1}^m L(E)^{g_i^h}$ is as follows.

- $\phi^{h[\nu^g(e)]}$ is a restriction of ϕ^h to $\{v \in \bigcup_{i=1}^m V(g_i^h) | \nu^g(e') = \nu^g(e), e' = (v, v'), v' \in \bigcup_{i=1}^m V(g_i^h)\}$.
- $G^{h[\nu^g(e)]}$ is a set of graphs $\{g_i^h[\nu^g(e)] | i = 1, \ldots, m\}$ where the root graph is the union of $r^h[\nu^g(e)]$ and graphs $\{g_i^h[\nu^g(e)] | \phi^{h[\nu^g(e)]}(v) \neq g_i^h[\nu^g(e)]$ for any $v \in \bigcup_{i=1}^m V(g_i^h[\nu^g(e)])\}$.

As the decomposition of graphs, $h[\mu^g(v)][\nu^g(e)]$ means $(h[\mu^g(v)])[\nu^g(e)]$, that is, the hierarchical graph is decomposed by the label $\mu^g(v)$ of a vertex v and then decomposed by the label $\nu^g(e)$ of an edge e. If h^s is a hierarchical subgraph of h, $h^s[\mu^g(v)]$ is a hierarchical subgraph of $h[\mu^g(v)]$.

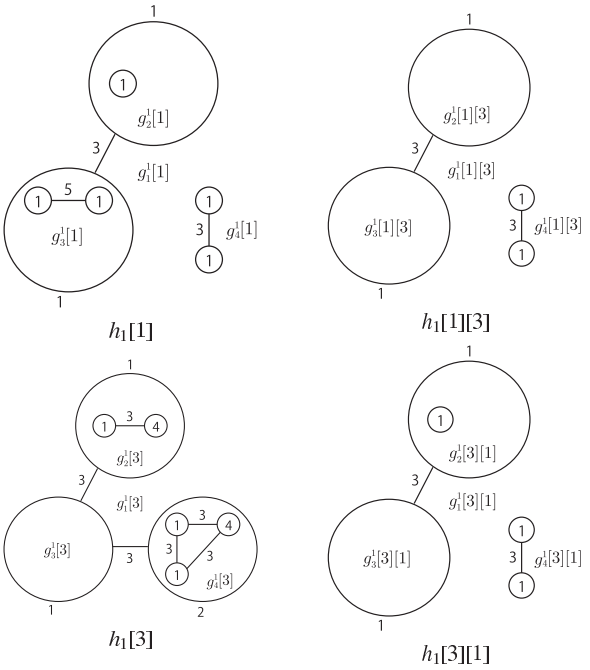

Fig. 2 Decomposed hierarchical graphs of h_1 of Fig. 1.

Example 3 **Figure 2** shows the decomposed hierarchical graphs of h_1 of Fig. 1. $h_1[1]$ and $h_1[3]$ are the decomposed ones according to the label 1 of a vertex and the label 3 of an edge, respectively. $h_1[1][3]$ and $h_1[3][1]$ are the decomposed ones of $h_1[1]$ and $h_1[3]$ according to the label 3 of an edge and the label 1 of a vertex, respectively. In both h_1 and h_2, the set of labels of vertices and the set of labels of edges are disjoint. The root graph of $h_1[1]$ is $g_1^1[1] \cup g_4^1[1]$. The hierarchical incidence matrix $D^1[3][1]$ of $h_1[3][1]$ is as follows.

$$D^1[3][1] = \begin{bmatrix} 1&0&3&1&0&0&0 \\ 0&1&3&0&0&0&0 \\ 3&3&3&0&0&0&0 \\ 1&0&0&1&0&0&0 \\ 0&0&0&0&1&0&3 \\ 0&0&0&0&0&1&3 \\ 0&0&0&0&3&3&3 \end{bmatrix}$$

We use the term "interlace" for two hierarchical graphs decomposed according to labels as follows .

Definition 10 A hierarchical graph $h_1 = (G^1, \phi^1)$ where $G^1 = \{g_i^1 | 1 \leq i \leq n\}$ *interlaces* another hierarchical graph $h_2 = (G^2, \phi^2)$, if the following conditions are satisfied.

- $h_1[\mu^g(v)]$ interlaces $h_2[\mu^g(v)]$ for each $\mu^g(v) \in \bigcup_{i=1}^n L(V)^{g_i^1}$
- $h_1[\mu^g(v)][\nu^g(e)]$ interlaces $h_2[\mu^g(v)][\nu^g(e)]$ for each $\mu^g(v) \in \bigcup_{i=1}^n L(V)^{g_i^1}$ and $\nu^g(e) \in \bigcup_{i=1}^n L(E)^{g_i^1}$

We can define it in a similar way for the other decomposed hierarchical graphs such as $h_1[\nu^g(e)]$, $h_2[\nu^g(e)]$, $h_1[\nu^g(e)][\mu^g(v)]$ and $h_2[\nu^g(e)][\mu^g(v)]$. It is clear that, if the eigenvalues of a decomposed hierarchical graph of h_1 do not interlace the eigenvalues of the corresponding decomposed hierarchical graph of h_2, h_1 is not a hierarchical subgraph of h_2. We denote a family of sets of eigenvalues of every decomposed hierarchical graph in the above definition as $AE(h)$.

5.3 Detecting Non Hierarchical Subgraphs by Eigenvalues

When we find hierarchical subgraphs of a query hierarchical

Algorithm 1 ConstructIndex(H)

Input: a hierarchical graph database H
Output: an index for H

```
1: create the root node of an index
2: for each h ∈ H do
3:     AE(h) ← ComputeEigenvalue(h)
4:     CreateNode(root,h,AE(h))
5: end for
```

graph, we construct the index as follows. When the eigenvalues of a hierarchical graph h_1 interlace the eigenvalues of another hierarchical graph h_2, the node which contains h_2 and its eigenvalues is a child of the node which contains h_1 and its eigenvalues. Given a query for this index, if the eigenvalues of the query do not interlace the eigenvalues of the hierarchical graph in a node, we know that all hierarchical graphs in the node and the descendant nodes are not hierarchical subgraphs of the query by the following proposition.

Proposition 2 Let α, β and γ be three sequences of real numbers. If γ interlaces β and γ does not interlace α, then β does not interlace α.

Proof Omitted.

α, γ and β correspond to sequences of eigenvalues of the following three hierarchical graphs, that is, a query, a hierarchical graph in a node *nod* of the index and a hierarchical graph in a child of *nod*, respectively.

6. Algorithms for Index Construction and Query Processing

We focus on finding hierarchical supergraphs of a query.

6.1 Index Construction

Algorithms 1, 2, 3 and 4 build an index by adding a new node for each hierarchical graph in a hierarchical graph database one by one from the root node. The root node is dummy and does not have any information. The addition of the new node is based on the way described in Section 5.1. When a node for a hierarchical graph h^p is a parent of a node for a hierarchical graph h^c, if all the following conditions are satisfied, a new node for a hierarchical graph h is inserted between the node for h^p and the node for h^c. That is, h interlaces h^p, h does not interlace h_c and h^c interlaces h. Each node of the tree except the root node stores the identifier of a hierarchical graph in a database and the eigenvalues of both the original hierarchical graph and its decomposed hierarchical graphs. The structure of a constructed index depends on the order of processing of hierarchical graphs in a database. We denote the hierarchical graph stored in a node *nod* of an interlace tree as h^{nod}.

Example 4 **Figure 3** demonstrates the index construction by Algorithm 1 for the hierarchical graphs h_1, h_2 and h_3. The eigenvalues of h_1, h_2 and h_3 have the following relations.

- $AE(h_1)$ does not interlace $AE(h_2)$ and $AE(h_3)$
- $AE(h_2)$ interlaces $AE(h_1)$ but does not interlace $AE(h_3)$
- $AE(h_3)$ interlaces $AE(h_1)$ but does not interlace $AE(h_2)$

The left index of Fig. 3 is constructed by processing in the order of h_2, h_3 and h_1. On the other hand, the right one is done in in the order of h_3, h_2 and h_1.

Algorithm 2 ComputeEigenvalue(h)

Input: a hierarchical graph $h = (G^h, \phi^h)$ where $G^h = \{g_i^h | 1 \le i \le n\}$
Output: a family $AE(h)$ of sets of eigenvalues of decomposed hierarchical graphs of h

```
1: for each label μ^g(v) ∈ ⋃_{i=1}^{n} L(V)^{g_i^h} do
2:     add Eig(h[μ^g(v)]) to AE(h)
3:     for each label ν^g(e) ∈ ⋃_{i=1}^{n} L(E)^{g_i^h} do
4:         add Eig(h[μ^g(v)][ν^g(e)]) to AE(h)
5:     end for
6: end for
7: return AE(h)
```

Algorithm 3 CreateNode(*nod*,*h*,*AE*(*h*))

Input: a node *nod* of an index, a hierarchical graph h and a family $AE(h)$ of sets of eigenvalues of decomposed hierarchical graphs of h
Output: an index

```
1: if nod has child nodes then
2:     for each child node cnod of nod do
3:         if CheckInterlace(h, h^cnod, AE(h), AE(h^cnod)) then
4:             CreateNode(cnod, h, AE(h))
5:             return
6:         else
7:             if CheckInterlace(h^cnod, h, AE(h^cnod), AE(h))) then
8:                 create a node for h as a child node of nod and a parent node
                   of cnod
9:                 store h and AE(h) in the node
10:                return
11:            end if
12:        end if
13:    end for
14: end if
15: create a node cnod for h as a child node of nod
16: store h and AE(h) in cnod
17: return
```

Algorithm 4 CheckInterlace($h_1, h_2, AE(h_1), AE(h_2)$)

Input: two hierarchical graphs $h_1 = (G^1, \phi^1)$ and $h_2 = (G^2, \phi^2)$ where $G^1 = \{g_i^1 | 1 \le i \le n\}$, two families $AE(h_1)$ and $AE(h_2)$ of sets of eigenvalues of decomposed hierarchical graphs of h_1 and h_2
Output: true when h_1 interlaces h_2, or false when h_1 does not interlace h_2

```
1: for each label μ^g(v) ∈ ⋃_{i=1}^{n} L(V)^{g_i^1} do
2:     for each label ν^g(e) ∈ ⋃_{i=1}^{n} L(E)^{g_i^1} do
3:         if Eig(h1[μ^g(v)][ν^g(e)]) ∈ AE(h1) does not interlace
           Eig(h2[μ^g(v)][ν^g(e)]) ∈ AE(h2) then
4:             return false
5:         end if
6:     end for
7: end for
8: for each label μ^g(v) ∈ ⋃_{i=1}^{n} L(V)^{g_i^1} do
9:     if Eig(h1[μ^g(v)]) ∈ AE(h1) does not interlace Eig(h2[μ^g(v)]) ∈ AE(h2)
       then
10:        return false
11:    end if
12: end for
13: return true
```

6.2 Query Processing

The query processing starts from the root node of the constructed index. Given a query q, for each child node *nod* of the

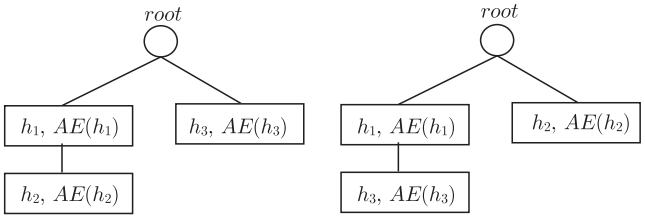

Fig. 3 Structures of indices constructed by proposed algorithms.

Algorithm 5 ProcessQuery(t, q)

Input: an index t constructed for a hierarchical graph database, a hierarchical graphs q as a query
Output: a set CH^q of answers to q

```
1: CheckNode(t, root, q)
```

Algorithm 6 CheckNode(t, nod, q)

Input: an index t, a node nod of the index, a query q
Output: a set CH^q of answers to q

```
1: for each child node cnod of nod do
2:     if FindCandidate(h^cnod, q, AE(h^cnod))) then
3:         add h^nod to CH^q
4:         CheckNode(t, cnod, q)
5:     end if
6: end for
7: return CH^q
```

Algorithm 7 FindCandidate($h_1, h_2, AE(h_1)$)

Input: two hierarchical graphs h_1 and h_2, and a family $AE(h_1)$ of sets of eigenvalues of decomposed hierarchical graphs of h_1
Output: true when h_1 interlaces h_2, or false when h_1 does not interlace h_2

```
1: for each label μ^g(v) of a vertex v of a graph g in G^h do
2:     for each label ν^g(e) of an edge e of a graph g in G^h do
3:         if Eig(h2[μ^g(v)][ν^g(e)]) does not interlace Eig(h1[μ^g(v)][ν^g(e)]) ∈
           AE(h1) then
4:             return false
5:         end if
6:     end for
7: end for
8: for each label μ^g(v) of a vertex v of a graph g in G^h do
9:     if Eig(h2[μ^g(v)]) does not interlace Eig(h1[μ^g(v)]) ∈ AE(h1) then
10:        return false
11:    end if
12: end for
13: return true
```

root, we check whether q interlaces a hierarchical graph h^{nod} in nod according to Definition 10 . If q interlaces h^{nod}, h^{nod} is returned as an answer to q, that is, a candidate for a hierarchical subgraph of q and then we check it for each child of nod. If q does not interlace h^{nod}, q is not a hierarchical subgraph of h^{nod} and the hierarchical graphs in the descendant nodes by Corollary 1. So it is not necessary to check it for the descendant nodes of nod. Algorithms 5, 6 and 7 show the detail of the query processing.

6.3 Maintenance of Index

When a new hierarchical graph h is added to the hierarchical graph database H, we add the new node for h to the index by the same procedure as constructing it. When a hierarchical graph h is deleted from H, we delete the node for h from the index. If the deleted node has child nodes, we connect the parent node with the child nodes. Since both the processes of addition and deletion are simple, we omit details.

7. Experimental Evaluation

In order to evaluate the proposed indexing method for hierarchical graphs, we show the time to construct the index, the time to process queries and the number of candidates for answers to queries on two kinds of hierarchical graphs, one of which is generated from real data and another is done artificially. The real data are originally a subset of Gene Ontology [17]. The artificial hierarchical graphs are prepared using a generator that we developed. We use the following procedure as a baseline in evaluation of query processing time since there is no appropriate indexing method for hierarchical graphs as a target for comparison so far to the authors' knowledge. That is, eigenvalues of hierarchical graphs in a database are computed and stored in advance. Given a hierarchical graph as a query, we compute the eigenvalues and compare them with the eigenvalues of each hierarchical graph in the database one by one based on the interlace theorem. We call this procedure Sequential Process.

It is difficult to decide exactly whether a number of hierarchical graphs are hierarchical subgraphs of other ones within a reasonable time and memory. Therefore we use a simpler procedure as a target for comparison in evaluating efficiency of the indexing method. That is, for each hierarchical graph, we ignore the hierarchical structure and check only whether each graph constituting it is a subgraph of a graph constituting another hierarchical graph in a combinatorial way. Algorithm 8 shows the detail of this procedure called Simple Search. Since Simple Search may be faster than procedures for the exact decision, it is valid to use it as a target for comparison instead of a procedure for the exact decision. In order to compare Simple Search adequately to the indexing method, we measure the time required to process hierarchical graphs which are not candidates detected using the index with Simple Search. We use 20 percent of them to compute the average time per a query in the experiments since it takes much time and memory to process all of the hierarchical graphs. We use VF2 [18] to decide whether a graph is a subgraph of another graph.

Table 1 shows the meanings of entries in tables where the results of experiments are shown. We develop all the software with Visual C++ 2012 and MATLAB 2012b on Windows 7 Professional 64 bit and use a PC with a 2.8 GHz Intel Core i3 processor and 16 GB RAM.

7.1 Hierarchical Graphs Generated from Real Data

We use Gene Ontology as a sample of real data. It is often used in experimental evaluation for graph based processing [19]. In this ontology, biological terms and relations among them are defined. We construct 1,356 hierarchical graphs by regarding biological terms as vertices of graphs and "is-a" relations among the biological terms as hierarchies of the graphs. A hierarchical graph consists of at most 50 graphs each of which has at most 5 vertices. On the assumption that ontologies are used to resolve contexts of terms, we connect the vertices of a graph with edges

Algorithm 8 SimpleSearch(H, q)

Input: a set H of hierarchical graphs, a hierarchical graph q as a query
Output: the set of candidates for answers to q

```
for each hierarchical graph h in H do
    for each graph g^q in the set G^q of graphs constituting q do
        if g^q is not a subgraph of any graph in the set G^h of graphs constituting h then
            add h to a set NC of hierarchical graphs
            break out of the inner loop
        end if
    end for
end for
return H \ NC
```

Table 1 Entries in tables.

Entry	Meaning
File Read	time [s] to read hierarchical graphs from an input file
Index Const.	time [s] to compute eigenvalues of indexed hierarchical graphs and construct the index
Eigen. Comp. (EC)	average time [ms] to decompose a query hierarchical graph and compute eigenvalues of all the decomposed hierarchical graphs
Tree Trav. (TT)	average time [ms] to detect candidates for answers to a query with the constructed index
Seq. Proc. (SP)	average time [ms] to detect candidates for answers to a query by comparing their eigenvalues based on the interlace theorem one by one without the constructed index (time to compute eigenvalues is not included)
Candidates (C)	average number of candidates for answers to a query
TT/SP	ratio of Tree Trav. (TT) to Seq. Proc. (SP)
(EC+TT)/(EC'+TT')	ratio of the processing time with the decomposition of hierarchical graphs to the time without it
C/C'	ratio of the number of candidates for answers with the decomposition of hierarchical graphs to the number without it
Simple Search	average time [ms] per query to process hierarchical graphs which are not candidates detected using the index with Algorithm 8
TT/Simple Search	ratio of Tree Trav. (TT) to Simple Search

Table 2 Index construction and query processing with proposed index and sequentially without index for real data.

Labels	3	4	5	6	7
File Read	1.23	1.32	1.34	1.42	1.53
Index Const.	0.85	0.92	0.97	1.04	1.14
Eigen. Comp. (EC)	0.6	0.7	0.7	0.9	0.8
Tree Trav. (TT)	0.8	0.9	0.9	0.9	1.0
Candidates (C)	294	218	172	139	114
Seq. Proc. (SP)	1.6	1.8	2.1	2.2	2.4
TT/SP	0.53	0.47	0.43	0.40	0.40
Simple Search	3.7	3.5	3.5	3.4	3.4
TT/Simple Search	0.23	0.24	0.26	0.26	0.29

and randomly add labels to them. In a context, all vertices have the same label. We choose the number of labels according to the number of vertices of a graph. Since indexing is usually useful for a large amount of data, the size of this ontology is not sufficient for evaluation. The purpose of this experiment is to show a preliminary result for data which are characteristic of the real world. We show the performance of the proposed indexing method for larger sizes of synthetic data later.

Table 2 shows the time required for constructing the index and processing queries, and the number of candidates for answers to queries in the case where the number of labels varies. The number of candidates to answers for a query is approximately from 8 to 21 percent of the number of hierarchical graphs in the data set. The query processing time (TT) with the index is from 40 to 53 percent of the time (SP) without the index. It is also from 23 to 29 percent of the processing time of Simple Search. As the number of labels assigned to vertices and edges decreases, the proposed method becomes effective relative to Simple Search since it is harder to check whether a graph is a subgraph of another graph in a combinatorial way. On the other hand, the proposed method becomes less effective relative to Sequential Process. It is thought to be due to structural differences of the constructed indices.

7.2 Hierarchical Graphs Generated Artificially

We use synthetic data to evaluate the performance of the proposed method for various sizes and types of hierarchical graphs. In our hierarchical graph generator, hierarchical structure among graphs that constitute a hierarchical graph is decided randomly. It is configured by the following parameters for a data set.

- number $|H|$ of hierarchical graphs in a data set H
- number $|G^h|$ of graphs that constitute a hierarchical graph $h \in H$
- number $|V^g|$ of vertices of each graph $g \in G^h$ that constitutes a hierarchical graph $h \in H$
- density $|E^g|/(|V^g|(|V^g|-1)/2)$ of each graph $g \in G^h$ that constitutes a hierarchical graph $h \in H$
- number $|L(V)^g|$ of labels assigned to vertices of each graph $g \in G^h$ that constitutes a hierarchical graph $h \in H$
- number $|L(E)^g|$ of labels assigned to edges of each graph $g \in G^h$ that constitutes a hierarchical graph $h \in H$

We generate a synthetic data set similar to the ontology. That is, different sizes of hierarchical graphs are included in a data set, and the number of smaller sized hierarchical graphs is more than the number of larger sized ones. In this experiment, 5 different sizes of hierarchical graphs are included in a data set. The size of a hierarchical graph depends on the number of graphs that constitute it and the number of their vertices. The number of hierarchical graphs of a size in a data set is inversely proportional to the size. For example, the number of hierarchical graphs consisting of two graphs is twice the number of hierarchical graphs consisting of four graphs. Positive integers are randomly assigned to vertices and edges of graphs as labels. Labels of vertices are different from labels of edges. We use all hierarchical graphs in each data set as queries. Therefore there is at least one answer for each query in the data set. The leftmost columns in Table 3 and 4 show the results on the same data set.

Table 3 (a) shows the results in the case where the number of hierarchical graphs in a data set varies from 9,300 to 21,700. In every data set, a hierarchical graph consists of 2, 4, 8, 16 or 32 graphs. The graph has 8 vertices and its density is 0.3. The number of labels assigned to the vertices and the edges is 8. In any data set, the average number of candidates for answers to a query is approximately 0.4 percent of the number of hierarchical graphs. As the number of hierarchical graphs in a data set increases, the processing time (TT) with the index increases but the both values of TT/SP and TT/Simple Search decrease. It means that the proposed index becomes effective relative to Simple Search and

Table 3 Index construction and query processing with proposed index and sequentially without index for artificial data.

(a) Varying number of hierarchical graphs

Hier. Graphs	9,300	12,400	15,500	18,600	21,700
File Read	144.8	199.7	268.8	356.9	411.3
Index Const.	105.3	148.2	204.1	277.3	319.5
Eigen. Comp. (EC)	4.2	4.1	4.1	4.3	4.1
Tree Trav. (TT)	15.5	18.8	22.6	27.1	30.0
Candidates (C)	36	50	59	72	83
Seq. Proc. (SP)	48.1	63.2	78.5	98.8	110.6
TT/SP	0.32	0.30	0.29	0.29	0.27
Simple Search	43.2	57.8	71.9	86.6	100.5
TT/Simple Search	0.35	0.32	0.31	0.31	0.29

(b) Varying number of graphs of which a hierarchical graph consists

Graphs	2-32	3-48	4-64	5-80	6-96
File Read	144.8	213.7	304.7	419.2	553.1
Index Const.	105.3	129.8	160.8	191.8	214.9
Eigen. Comp. (EC)	4.2	5.6	7.0	8.5	10.3
Tree Trav. (TT)	15.5	17.6	19.2	20.3	21.6
Candidates (C)	36	28	23	20	18
Seq. Proc. (SP)	48.1	49.0	49.4	49.1	48.9
TT/SP	0.33	0.38	0.42	0.44	0.44
Simple Search	43.2	60.0	78.3	94.0	115.2
TT/Simple Search	0.35	0.29	0.25	0.22	0.19

(c) Varying number of vertices of graphs in a hierarchical graph

Vertices	8	10	12	14	16
File Read	144.8	296.7	300.9	429.1	607.6
Index Const.	105.3	141.9	157.5	194.3	222.8
Eigen. Comp. (EC)	4.2	5.8	7.2	9.0	11.3
Tree Trav. (TT)	15.5	18.1	19.0	20.9	21.4
Candidates (C)	36	25	20	17	15
Seq. Proc. (SP)	48.1	48.6	48.9	48.8	49.5
TT/SP	0.32	0.37	0.39	0.43	0.43
Simple Search	43.2	47.4	53.0	57.6	62.2
TT/Simple Search	0.35	0.38	0.36	0.36	0.34

(d) Varying density of graphs in a hierarchical graphs

Density	0.30	0.40	0.50	0.60	0.70
File Read	144.8	170.1	200.8	233.9	263.0
Index Const.	105.3	117.2	132.4	148.0	154.7
Eigen. Comp. (EC)	4.2	4.7	5.4	6.1	6.5
Tree Trav. (TT)	15.5	16.6	17.8	18.7	18.9
Candidates (C)	36	30	25	23	21
Seq. Proc. (SP)	48.1	47.4	49.0	48.9	48.5
TT/SP	0.32	0.35	0.36	0.38	0.39
Simple Search	43.2	43.9	44.0	44.7	44.9
TT/Simple Search	0.35	0.38	0.40	0.42	0.42

(e) Varying number of labels assigned to vertices and edges of graphs in hierarchical graphs

Labels	3	4	5	6	7
File Read	95.4	144.8	219.2	313.3	435.2
Index Const.	56.0	105.3	180.1	274.8	396.0
Eigen. Comp. (EC)	3.1	4.2	5.4	6.8	8.3
Tree Trav. (TT)	7.2	15.5	26.8	40.9	57.5
Candidates (C)	46	36	30	25	23
Seq. Proc. (SP)	25.1	48.1	73.0	102.1	133.9
TT/SP	0.29	0.32	0.37	0.40	0.43
Simple Search	50.8	43.2	40.8	62.0	39.1
TT/Simple Search	0.14	0.35	0.66	0.66	1.47

Sequential Process.

Table 3 (b) shows the results in the case where the set of the number of graphs that constitute each hierarchical graph in a data set varies from { 2, 4, 8, 16, 32 } to { 6, 12, 24, 48, 96 }. In the first row of the table, 2-32 and 3-48 mean the sets { 2, 4, 8, 16, 32 } and { 3, 6, 12, 24, 48 }, respectively, for example. In each data set, the number of hierarchical graphs is 9,300. The number of vertices of each graph that constitutes a hierarchical graph is 8 and the density of the graph is 0.3. The number of labels assigned to vertices and edges of the graph is 8. The average number (CC) of candidates for answers to a query is approximately from 0.2 to 0.4 percent of the number of hierarchical graphs in the data sets. As the number of graphs that constitute a hierarchical graph increases, the average number (CC) of candidates and TT/Simple Search decreases. This means that the proposed index becomes effective. On the other hand, the value of TT/SP increases. This is due to the increase in the number of eigenvalues of decomposed hierarchical graphs in each node of the constructed index, which are compared in processing a query.

Table 3 (c) shows the results in the case where the number of vertices of each graph that constitutes hierarchical graphs in a data set varies from 8 to 16. In each data set, the number of hierarchical graphs is 9,300. A hierarchical graph consists of 2, 4, 8, 16 or 32 graphs. The density of the graph is 0.3. The number of labels assigned to the vertices and the edges is 8. Table 3 (d) shows the results in the case where the density of each graph that constitutes hierarchical graphs in a data set varies from 0.3 to 0.7. In each data set, the number of hierarchical graphs is 9,300. A hierarchical graph consists of 2, 4, 8, 16 or 32 graphs. The graph has 8 vertices. The number of labels assigned to the vertices and the edges is 8. In both experiments, the average number of candidates for answers to a query is approximately from 0.2 to 0.4 percent of the number of hierarchical graphs in the data sets. It decreases as the size of graphs, that is, the number of the vertices or their density, that constitute a hierarchical graph increases. On the other hand, the value of TT/SP increases due to the increase in the number of eigenvalues of decomposed hierarchical graphs similar to Table 3 (b). The observed variation in TT/Simple Search of Table 3 (c) and Table 3 (d) may be due to VF2.

Table 3 (e) shows the results in the case where the number of labels assigned to vertices and edges of each graph that constitute hierarchical graphs varies from 3 to 7. In each data set, the number of hierarchical graphs is 9,300. A hierarchical graph consists of 2, 4, 8, 16 or 32 graphs. The graph has 8 vertices and its density is 0.3. The average number of candidates for answers to a query is approximately from 0.2 to 0.5 percent of the number of hierarchical graphs in the data sets. As the number of labels assigned to vertices and edges of each graph that constitute hierarchical graphs increases, it decreases since a hierarchical graph

Table 4 Index construction and query processing without hierarchical graph decomposition for artificial data.

(a) Varying sizes of graphs in a hierarchical graph

Graphs	2-32	3-48	4-64	5-80	6-96
File Read	141.8	213.7	304.7	419.2	553.1
Index Const.	29.1	61.2	105.1	167.2	252.5
Eigen. Comp. (EC')	2.7	5.5	10.1	16.3	24.7
Tree Trav. (TT')	4.5	7.8	9.5	14.1	15.7
Candidates (C')	1,618	1,604	1,571	1,550	1,537
(EC+TT)/(EC'+TT')	2.72	1.75	1.33	0.95	0.79
C/C'	0.02	0.02	0.01	0.01	0.01

(b) Varying number of vertices of graphs in a hierarchical graph

Vertices	8	10	12	14	16
File Read	141.8	296.7	300.9	429.1	607.6
Index Const.	29.1	58.9	104.7	171.5	283.6
Eigen. Comp. (EC')	2.7	5.4	9.9	17.4	28.5
Tree Trav. (TT')	4.5	7.7	9.6	14.3	15.5
Candidates (C')	1,618	1,660	1,653	1,655	1,033
(EC+TT)/(EC'+TT')	2.72	1.83	1.35	0.95	0.74
C/C'	0.02	0.02	0.01	0.01	0.01

has lower chances of being a hierarchical subgraph of another hierarchical graph. The query processing time (TT) with the index increases. It is due to structural differences of the constructed indices and increase of the time required to decompose a query hierarchical graph according to labels. The reason why the processing time (SP) without the index increases is also that the time required to decompose a query hierarchical graph increases. On the other hand, Simple Search becomes faster since sizes of hierarchical graphs do not change and it is easier to check whether a graph is a subgraph of another graph. File Read increases since the size of an input data increases as the number of digits of labels assigned to vertices and edges of graphs increases.

7.3 Effect of Hierarchical Graph Decomposition

Here we evaluate the effect of the decomposition method of hierarchical graphs according to labels assigned to vertices and edges of graphs, which is described in Section 5.2. We compare the query processing time and the number of candidates for answers to a query in the case with the decomposition to those in the case without it. The settings and data sets of the experiments whose results are shown in **Table 4** (a) and Table 4 (b) are the same as in Table 3 (b) and Table 3 (c), respectively.

Table 4 (a) shows the results in the case where the set of the number of graphs that constitute a hierarchical graph in a data set varies from { 2, 4, 8, 16, 32 } to { 6, 12, 24, 48, 96 }. Table 4 (b) shows the results in the case where the number of vertices of each graph that constitutes hierarchical graphs in the data set varies from 8 to 16. In both experiments, the average number of candidates for answers to a query is approximately from 50 to 100 times as many as the average number of candidates in the case with the decomposition as will be noted from the value C/C' in the tables. They show that the proposed decomposition method is very effective in reducing the number of candidates. Decreasing the values of (EC+TT)/(EC'+TT') in the tables shows that the average time for processing a query in the case with decomposition becomes faster than that without the decomposition as the sizes of graphs that constitute a hierarchical graph become larger.

8. Conclusion

We propose an indexing method for hierarchical graphs, which is based on the relation of interlacing sequences of eigenvalues. Given a hierarchical graph as a query, we can filter hierarchical graphs which do not contain it as substructures and find the candidates for answers to the query with the index. We also define a matrix representation of a hierarchical graph. In order to improve the filtering performance of the proposed index and reduce the cost of computing eigenvalues, we propose the decomposition of hierarchical graphs according to labels assigned to vertices and edges. By experimental evaluation, we show the effectiveness of the proposed method. Since graphs are hierarchical graphs without hierarchy, our approach is applicable to them. The proposed methods may be improved by combining with suitable combinatorial methods.

References

[1] Cook, D.J. and Holder, L.B. (Eds.): *MINING GRAPH DATA*, John Wiley & Sons, Inc. (2007).

[2] Giacomo, E., Didimo, W., Grilli, L. and Liotta, G.: WhatsOnWeb: Using Graph Drawing to Search the Web, *Graph Drawing*, Healy, P. and Nikolov, N. (Eds.), Lecture Notes in Computer Science, Vol.3843, Springer Berlin Heidelberg, pp.480–491 (2006).

[3] Birchall, K. and Gillet, V.J.: Reduced Graphs and Their Applications in Chemoinformatics, *Chemoinformatics and Computational Chemical Biology*, Methods in Molecular Biology, Vol.672, Humana Press, chapter 8, pp.197–212 (2011).

[4] Haemers, W.H.: Interlacing Eigenvalues and Graphs, *Linear Algebra and its Applications*, Vol.226-228, pp.593–616 (1995).

[5] Yan, X. and Han, J.: Graph Indexing, *Managing and Mining Graph Data*, Aggarwal, C.C. and Wang, H. (Eds.), Springer, chapter 5, pp.161–180 (2010).

[6] Shokoufandeh, A., Macrini, D., Dickinson, S., Siddiqi, K. and Zucker, S.W.: Indexing Hierarchical Structures Using Graph Spectra, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.27, No.7, pp.1125–1140 (2005).

[7] Demirci, M.F., van Leuken, R.H. and Veltkamp, R.C.: Indexing through laplacian spectra, *Computer Vision and Image Understanding*, Vol.110, No.3, pp.312–325 (2008).

[8] Zhang, N., Özsu, M.T., Ilyas, I.F. and Aboulnaga, A.: FIX: Feature-based Indexing Technique for XML Documents, *Proc. 32nd International Conference on Very Large Data Bases*, VLDB Endowment, pp.259–270 (2006).

[9] Zou, L., Chen, L., Yu, J.X. and Lu, Y.: A novel spectral coding in a large graph database, *Proc. 11th International Conference on Extending Database Technology: Advances in Database Technology*, pp.181–192 (2008).

[10] Shokoufandeh, A., Dickinson, S.J., Siddiqi, K. and Zucker, S.W.: Indexing using a spectral encoding of topological structure, *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (1999).

[11] Cheng, J., Ke, Y., Ng, W. and Lu, A.: Fg-index: Towards verification-free query processing on graph databases, *Proc. 2007 ACM SIGMOD International Conference on Management of Data*, pp.857–872 (2007).

[12] Yan, X. and Han, J.: gSpan: Graph-Based Substructure Pattern Mining, *Proc. IEEE International Conference on Data Mining*, pp.721–724 (2002).

[13] Cheng, J., Ke, Y. and Ng, W.: Efficient query processing on graph databases, *ACM Trans. Database Syst.*, Vol.34, No.1, pp.2:1–2:48 (2009).

[14] Shasha, D., Wang, J.T.L. and Giugno, R.: Algorithmics and applications of tree and graph searching, *Proc. 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp.39–52 (2002).

[15] He, H. and Singh, A.K.: Closure-Tree: An Index Structure for Graph Queries, *Proc. 22nd International Conference on Data Engineering*, p.38 (2006).

[16] Katayama, K., Amagasa, Y. and Nagaya, H.: Detecting Non-subgraphs Efficiently by Comparing Eigenvalues of Decomposed

Graphs, *IEICE Trans. Information and Systems*, Vol.E95-D, No.11, pp.2724–2727 (2012).
[17] the Gene Ontology Consortium: Gene Ontology: Tool for the unification of biology, *Nature Genetics*, Vol.25, pp.25–29 (2000).
[18] Cordella, L.P., Foggia, P., Sansone, C. and Vento, M.: An improved algorithm for matching large graphs, *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pp.149–159 (2001).
[19] He, H. and Singh, A.K.: Graphs-at-a-time: Query language and access methods for graph databases, *Proc. 2008 ACM SIGMOD International Conference on Management of Data*, pp.405–418 (2008).

Kaoru Katayama received his Ph.D. degree in informatics from Kyoto University in 2000. He is currently an associate professor in the Graduate School of Information and Communication Systems, Tokyo Metropolitan University. His research interests are in the areas of data engineering and data mining.

Ernest Weke Maina received his M.E. degree in Electrical Engineering from Tokyo Metropolitan University in 1996. He is currently a freelance software developer. His research interests are in the areas of graph and network algorithms.