IEICE TRANS. INF. & SYST., VOL.E92–D, NO.2 FEBRUARY 2009

PAPER Special Section on Foundations of Computer Science

A Polynomial Time Algorithm for Finding a Minimally Generalized Linear Interval Graph Pattern

Hitoshi YAMASAKI[†a)], *Nonmember and* **Takayoshi SHOUDAI**[†], *Member*

SUMMARY A graph is an interval graph if and only if each vertex in the graph can be associated with an interval on the real line such that any two vertices are adjacent in the graph exactly when the corresponding intervals have a nonempty intersection. A number of interesting applications for interval graphs have been found in the literature. In order to find structural features common to structural data which can be represented by intervals, this paper proposes new interval graph structured patterns, called linear interval graph patterns, and a polynomial time algorithm for finding a minimally generalized linear interval graph pattern explaining a given finite set of interval graphs.

key words: *interval graphs, PQ-trees, graph structured patterns, graph mining, computational learning theory*

1. Introduction

A graph $G = (V, E)$ is an *interval graph* if and only if for each vertex $v \in V$, a closed interval I_v in the real line can be associated such that for each pair of vertices $u, v \in V$ $(u \neq v)$, $(u, v) \in E$ if and only if $I_u \cap I_v \neq \emptyset$. For example, in Fig. 1, G is an interval graph which has its interval representation $R(G)$. One important application of interval graphs is a physical mapping in genome research, that is, to reconstruct the relative positions of fragments of DNA along the genome from certain pairwise overlap information [13]. Reliable and complete overlap information is very costly and practically not available. Probe interval graphs were introduced by Zhang et al.[8], [14] to represent only partial overlap information. As another application, the mutual exclusion scheduling problem is known to be formalized by a subclass of interval graphs [3].

In order to represent interval patterns common to interval structured data, we propose *interval graph patterns* which consist of interval graph structures and simplicial variables. The formal definition is described in Sect. 2. An interval graph pattern is called *linear* if all variables in it have mutually distinct variable labels. For an interval graph pattern g, the interval graph language of g, denoted by $L(g)$, is the set of all interval graphs which are obtained from g by substituting arbitrary interval graphs for all variables in g. In Fig. 1, f is a linear interval graph pattern with three variables of variable labels x, y and z, and $R(f)$ is an interval representation of f. The interval graph G is obtained from f by replacing x, y and z with g_1, g_2 and g_3, respectively. Then $G \in L(f)$. For a finite set of interval graphs S, a *minimally generalized linear interval graph pattern* explaining S is defined as a linear interval graph pattern g such that $S \subseteq L(g)$ and there exists no linear interval graph pattern g' satisfying $S \subseteq L(g') \subsetneq L(g)$. In this paper, we give a polynomial time algorithm for finding a minimally generalized linear interval graph pattern explaining a given finite set of interval graphs.

It is known that the general problem of deciding graph isomorphism appears to be hard. However for some special classes of graphs, isomorphism can be decided efficiently. The class of interval graphs is the case. Lueker and Booth [6] gave a linear time algorithm for interval graph isomorphism. Their isomorphism algorithm uses a data structure called a *labeled PQ-tree* which is an extension of *PQ-tree* [2]. PQ-trees are used to represent the permutations of a set in which specified subsets of the set occur consecutively. In this paper, we introduce a new tree structured pattern, called a PQ-tree pattern, to design polynomial time algorithms for interval graph patterns.

As related works, Suzuki et al.[11] and Matsumoto et al.[7] showed polynomial time learnabilities of linear tree patterns with internal structured variables by using various learning models. Takami et al.[12] gave a polynomial time learning algorithm for graph structured patterns based on two-terminal series parallel (TTSP) graphs, which are used as data models in applications for electric networks and scheduling problems. Many chemical compounds are known to be represented by outerplanar graphs. Horváth et al.[5] developed a frequent subgraph mining algorithm in a class of outerplanar graphs that works in incremental polynomial time. Sasaki et al.[10] presented an effective algorithm of enumerating all frequent block preserving outerplanar graph patterns from a given finite set of outerplanar graphs.

This paper is organized as follows. In Sect. 2, we formally define interval graph patterns and PQ-tree patterns. In Sect. 3, we present a polynomial time matching algorithm which decides whether or not a given interval graph pattern matches a given interval graph. In Sect. 4, we present a polynomial time algorithm for finding a minimally generalized linear interval graph pattern explaining a given finite set of interval graphs. In Sect. 5, we conclude this paper with future works.

2. Interval Graph Patterns

For a graph $G = (V, E)$ and a vertex $u \in V$, the set of vertices

Manuscript received March 31, 2008.
Manuscript revised July 02, 2008.

[†]The authors are with the Department of Informatics, Kyushu University, Fukuoka-shi, 819–0395 Japan.
a) E-mail: h-yama@i.kyushu-u.ac.jp

DOI: 10.1587/transinf.E92.D.120

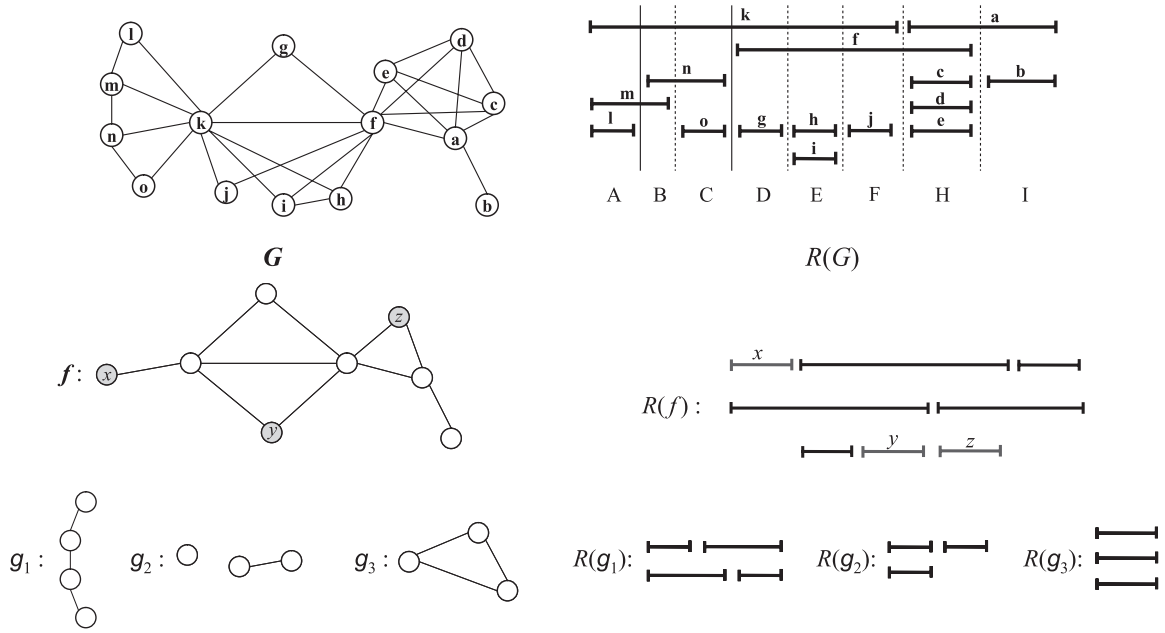

Fig. 1 An interval graph G (its interval representation $R(G)$) and an interval graph pattern f (its interval representation $R(f)$). The interval graph G is the instance of f by θ where $\theta = [x/g_1, y/g_2, z/g_3]$.

adjacent to u, called the *neighborhood* of u, is denoted by $N_G(u)$. Occasionally we call $N_G(u)$ the *open* neighborhood of u and $N_G(u) \cup \{u\}$ the *closed* neighborhood of u. We denote the *closed* neighborhood of u by $N_G[u]$. A clique of a graph G is a complete subgraph of G. A vertex u is *simplicial* if the subgraph induced by $N_G(u)$ is a clique.

Definition 1: Let $G = (V, E)$ be an interval graph. Let V_g and H_g be a partition of V such that $V_g \cup H_g = V$ and $V_g \cap H_g = \emptyset$. A triplet $g = (V_g, E, H_g)$ is called an *interval graph pattern* if all vertices in H_g are simplicial and no two vertices in H_g are adjacent. Let X be an infinite alphabet. We call an element in X a *variable label*. Each variable in an interval graph pattern is labeled with a variable label in X. We call elements in V_g and H_g a vertex and a *variable*, respectively.

For any interval graph pattern g, we denote the sets of vertices, edges, and variables by $V(g)$, $E(g)$, and $H(g)$, respectively. The *size* of g is defined as $|V(g)|+|H(g)|$. For any $u \in H(g)$, we denote the variable label of u by $x(u)$. An interval graph pattern g is called *linear* if all variables in $H(g)$ have mutually distinct variable labels in X, that is, for any two variables $u, v \in H(g)$, $x(u) \neq x(v)$. We denote the set of all interval graphs by $\mathcal{IG}$, the set of all interval graph patterns by $\mathcal{IGP}$, and the set of all linear interval graph patterns by $\mathcal{LIGP}$.

Definition 2: Let f and g be two interval graph patterns in $\mathcal{IGP}$. We say that f and g are *isomorphic*, denoted by $f \cong g$, if there exists a bijection $\varphi : V(f) \cup H(f) \to V(g) \cup H(g)$ such that (1) for any $u, v \in V(f) \cup H(f)$, $(u, v) \in E(f)$ if and only if $(\varphi(u), \varphi(v)) \in E(g)$, (2) $v \in V(f)$ if and only if $\varphi(v) \in V(g)$, and (3) for any $u, v \in H(f)$, $x(u) = x(v)$ if and only if $x(\varphi(u)) = x(\varphi(v))$.

Definition 3: Let g_1 and g_2 be interval graph patterns in $\mathcal{IGP}$. For a variable label $x \in X$, the form x/g_2 is called a *binding* for x. A new interval graph pattern $g_1[x/g_2]$ is obtained from g_1 and x/g_2 by connecting all vertices and variables in $V(g_2) \cup H(g_2)$ to all vertices in $N_{g_1}(h)$ for each variable h such that $x(h) = x$, and then removing h from g_1. Formally $g_3 = g_1[x/g_2]$ is defined as $V(g_3) = V(g_1) \cup V(g_2)$, $E(g_3) = E(g_1) \cup E(g_2) \cup \{(u, v) \mid u \in N_{g_1}(h), v \in V(g_2) \cup H(g_2)\} - \{(u, h) \mid u \in N_{g_1}(h)\}$, and $H(g_3) = H(g_1) \cup H(g_2) - \{h\}$. A *substitution* θ is a finite collection of bindings $[x_1/g_1, \ldots, x_n/g_n]$, where x_i's are mutually distinct variable labels in X.

The interval graph pattern $f\theta$, called the *instance* of f by θ, is obtained by applying all the bindings x_i/g_i on f simultaneously. We give an example of a substitution in Fig. 1. For an interval graph G and an interval graph pattern g, we say that g *matches* G if there exists a substitution θ such that $G \cong g\theta$.

Definition 4 ([6]): A *labeled PQ-tree* is a node-labeled ordered tree whose internal nodes consist of two classes, namely P-nodes and Q-nodes. Each leaf and P-node is labeled with a nonnegative integer ℓ and each Q-node with m children is labeled with a lexicographically sorted sequence of m' pairs of integers $(i_1, j_1), \ldots, (i_{m'}, j_{m'})$ where $m' \geq 1$ and $1 \leq i_k \leq j_k \leq m$ for $k = 1, 2, \ldots, m'$. We denote the label of a node a by $Label(a)$. We say that two labeled PQ-trees T_1 and T_2 are equivalent, denoted by $T_1 \equiv T_2$, if T_2 is obtained from T_1 by applying any combination of the following transformations:

(a) arbitrarily reordering the children of a P-node, and
(b) reversing the ordering of the m children of a Q-node and replacing the label $(i_1, j_1), \ldots, (i_{m'}, j_{m'})$ with the

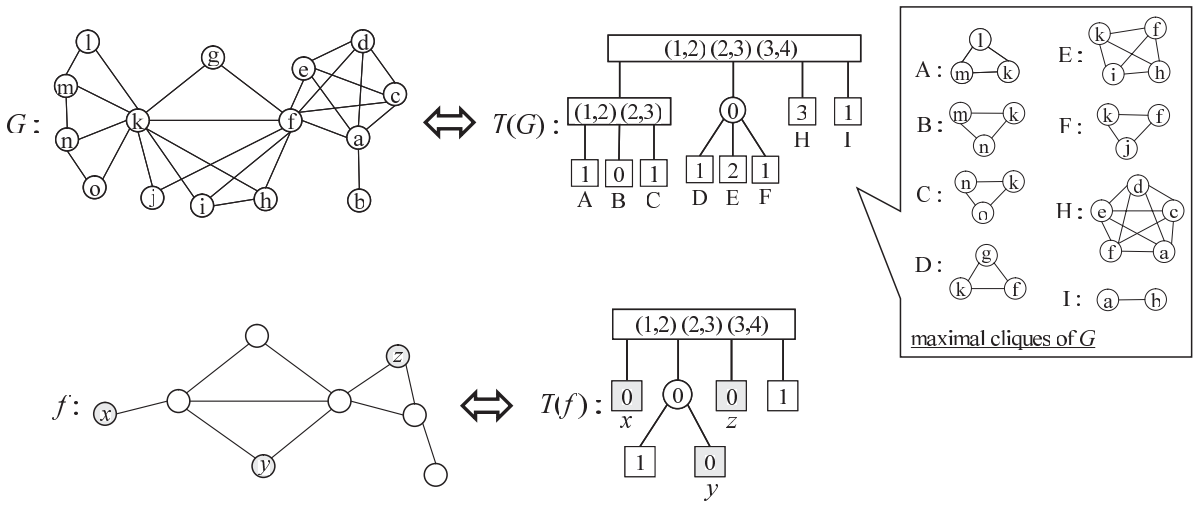

Fig. 2 An interval graph G has 8 maximal cliques (see also Fig. 1). Each maximal clique corresponds to one of leaves of $T(G)$. $T(f)$ is the PQ-tree pattern of a linear interval graph pattern f (Definition 5). In this and subsequent figures, P-nodes are drawn as circles, Q-nodes and leaves as rectangles, and variables as dot-filled rectangles.

lexicographically sorted sequence of $(m + 1 - j_1, m + 1 - i_1), \ldots, (m + 1 - j_{m'}, m + 1 - i_{m'})$.

For a node a of a PQ-tree T, we denote by $T[a]$ the subtree induced by a and all descendants of a. For a Q-node a and its label $Label(a)$, we denote the label after applying the transformation (b) to a by $Label^r(a)$. The *frontier* of a PQ-tree T is the ordering of its leaves obtained by reading them from left to right. The frontier of a node a, denoted by $F(a)$, is the frontier of $T[a]$. An ordering of the leaves of T is *consistent* with T if it is the frontier of a PQ-tree equivalent to T. For each vertex u of a graph G, let $C(u)$ be the set of maximal cliques which contain u. It is known that G is an interval graph if and only if there exists a linear ordering of all maximal cliques of G such that for each vertex u of G, the elements of $C(u)$ appear consecutively within the ordering. Lueker and Booth [6] gave a linear time algorithm, given an interval graph G, to construct a labeled PQ-tree T so that there is a bijection ψ from the set of all leaves of T to the set of all maximal cliques of G satisfying the conditions (1)–(3).

(1) Let k be the number of leaves of T. An ordering $(b_1, \ldots, b_k)$ of the leaves of T is consistent with T if and only if for any vertex $u \in V(G)$, an element of $C(u)$ appears consecutively in $(\psi(b_1), \ldots, \psi(b_k))$.
(2) For any vertex $u \in V(G)$, the *characteristic node* of u is the deepest node a in T such that $F(a)$ contains all elements of $\psi^{-1}(C(u))$. For any leaf or P-node a, $Label(a) = |\{u \in V(G) \mid a \text{ is the characteristic node of } u\}|$.
(3) For any Q-node a and its children $c_1, \ldots, c_m$, $Label(a)$ contains (i, j) $(1 \le i \le j \le m)$ if and only if there is a vertex $u \in V(G)$ such that $\psi^{-1}(C(u))$ is the set of all leaves in the frontiers of $c_i, \ldots, c_j$.

We denote the labeled PQ-tree obtained from an interval graph G by $T(G)$. For a labeled PQ-tree T, we denote the sets of nodes and edges by $V(T)$ and $E(T)$, respectively.

A labeled PQ-tree $T(G)$ is a tree representation of an interval representation $R(G)$ of an interval graph G. For example, the labeled PQ-tree of the interval representation $R(G)$ in Fig. 1 is described in Fig. 2. A path from the root to a leaf of $T(G)$ corresponds to one of the segments of $R(G)$. For example, the paths from the root to the leaves A–F,H,I of $T(G)$ in Fig. 2 corresponds to the segments A–F,H,I of $R(G)$ in Fig. 1, respectively. The paths also correspond to the maximal cliques of G. One of the correspondences is described in the upper right frame Fig. 2.

Theorem 1 ([6]): For interval graphs G_1 and G_2, $G_1 \cong G_2$ if and only if $T(G_1) \equiv T(G_2)$.

Definition 5: Let g be an interval graph pattern and $G = (V(g) \cup H(g), E(g))$. The *PQ-tree pattern* of g is the labeled PQ-tree with variables, denoted by $T(g)$, which is obtained from $T(G)$ by, for all characteristic nodes $a \in V(T(G))$ of the variables $h \in H(g)$, decreasing the label of a by one and attaching the variable label of h to a as its variable label. We note that the characteristic node of any variable is a leaf in $T(G)$ since all variables are simplicial in g.

A PQ-tree pattern is called *linear* if all variables in it have mutually distinct variable labels. We give an example of an interval graph pattern f and its PQ-tree pattern $T(f)$ in Fig. 2. We denote the set of all labeled PQ-trees by $\mathcal{PQT}$, the set of all PQ-tree patterns by $\mathcal{PQTP}$, and the set of all linear PQ-tree patterns by $\mathcal{LPQTP}$. We remind that if v is a variable or a P-node, then $Label(v)$ is a nonnegative integer, otherwise, a sequence of pairs of positive integers.

Definition 6: Let t_1 and t_2 be PQ-tree patterns and r the root of t_2. We assume that if a PQ-tree pattern consists of a single node, the label of the node is a positive integer. Let h be a variable of t_1 whose variable label is $x \in X$. Let x/t_2 be a binding for x. A new PQ-tree pattern $t_1[x/t_2]$ is

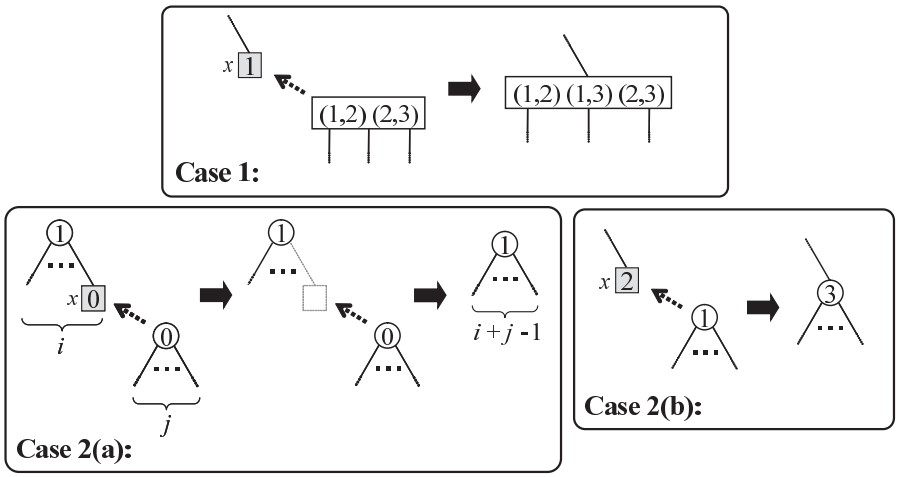

Fig. 3 Replacements of PQ-trees in Definition 6.

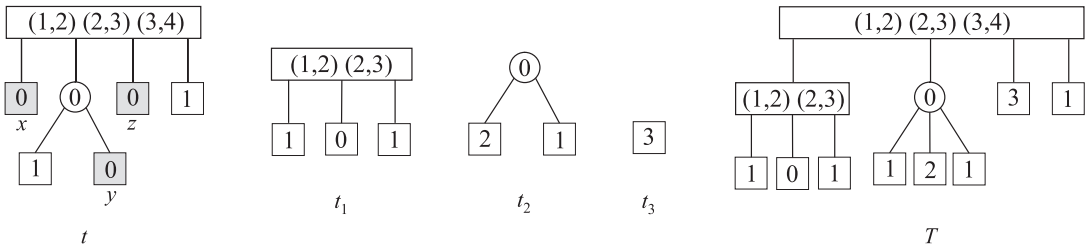

Fig. 4 t_1, t_2 and t_3 are the labeled PQ-trees of g_1, g_2 and g_3 in Fig. 1, respectively. And t is the PQ-tree patterns of f in Fig. 1. T is the instance of t by a substitution $\tau = [x/t_1, y/t_2, z/t_3]$.

obtained from t_1 by applying x/t_2 to t_1 in the following way (see Fig. 3).

1. If r is a Q-node with k children, r is identified with h (the new node is a Q-node) and its label is the lexicographically sorted sequence of the concatenation of $Label(r)$ and $\underbrace{(1,k),\ldots,(1,k)}_{Label(h)\text{ times}}$.
2. If either r is a P-node or t_2 is a single node, and
   a. if $Label(r) = Label(h) = 0$ and the parent of h is a P-node, the children of r are directly connected to the parent of h (h and r are removed),
   b. otherwise, r is identified with h (the new node is a P-node) and its label is $Label(r) + Label(h)$.

A finite collection of bindings $\tau = [x_1/t_1, \ldots, x_n/t_n]$ is called a *substitution*, where x_i's are mutually distinct variable labels in X.

For example, in Fig. 4, the bindings x/t_1, y/t_2 and z/t_3 are the cases (1), (2-a) and (2-b), respectively. The PQ-tree pattern $t\tau$, called the *instance* of t by τ, is obtained by applying all the bindings x_i/t_i on t simultaneously. We have the next lemma for an interval graph pattern and a PQ-tree pattern.

Lemma 1: Let f and g be interval graph patterns and x a variable label of a vertex in $H(f)$. Then, $T(f)[x/T(g)] \equiv T(f[x/g])$.

Proof. Let g_1 and g_2 be interval graph patterns and h a variable in $H(g_1)$. Let $g_3 = g_1[x(h)/g_2]$. From Definition 3, for each vertex or variable $u \in V(g_2) \cup H(g_2)$ and each vertex $v \in N_{g_1}(h)$, we can show that $N_{g_3}[u] \subseteq N_{g_3}[v]$. From this fact, it is easy to see that the characteristic node of u is a descendant of v in the PQ-tree pattern g_3. Therefore we can see that the PQ-tree pattern $T(g_1)[x/T(g_2)]$ represents connections between g_1 and g_2 produced by the substitution. Additionally, we note that there exists no P-node which has a P-node labeled with 0 as a child in any PQ-tree, because, for such a PQ-tree T, there exists an ordering of all maximal cliques of an interval graph G corresponding T such that, for every vertex v of G, the maximal cliques containing vertex v occur consecutively, and the ordering is not consistent with T. Definition 6 includes the method of a substitution which does not generate such a PQ-tree pattern, so $T(g_1)[x/T(g_2)]$ can represent all orderings of maximal cliques of g_3 which have the consecutive property. □

3. Interval Graph Pattern Matching

For a PQ-tree T and a PQ-tree pattern t, we say that t *matches* T if there exists a substitution τ such that $T \equiv t\tau$. Unfortunately the general interval graph pattern matching problem is hard to compute.

Theorem 2: The problem of deciding, given an interval graph pattern $g \in \mathcal{IGP}$ and an interval graph $G \in \mathcal{IG}$,

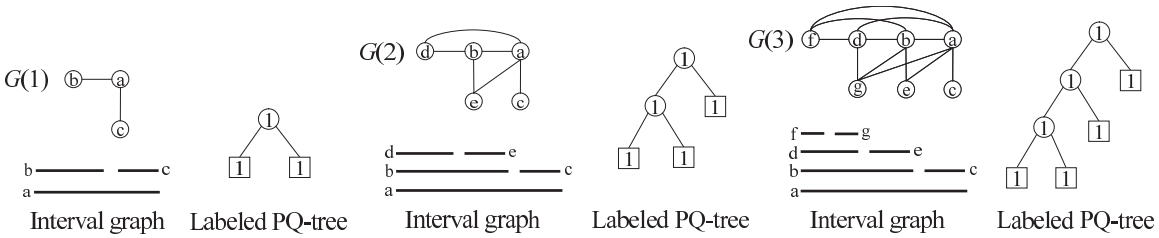

Fig. 5 The first three interval graphs and their labeled PQ-trees which represent the digits in $V = \{1, 2, \ldots, |V|\}$.

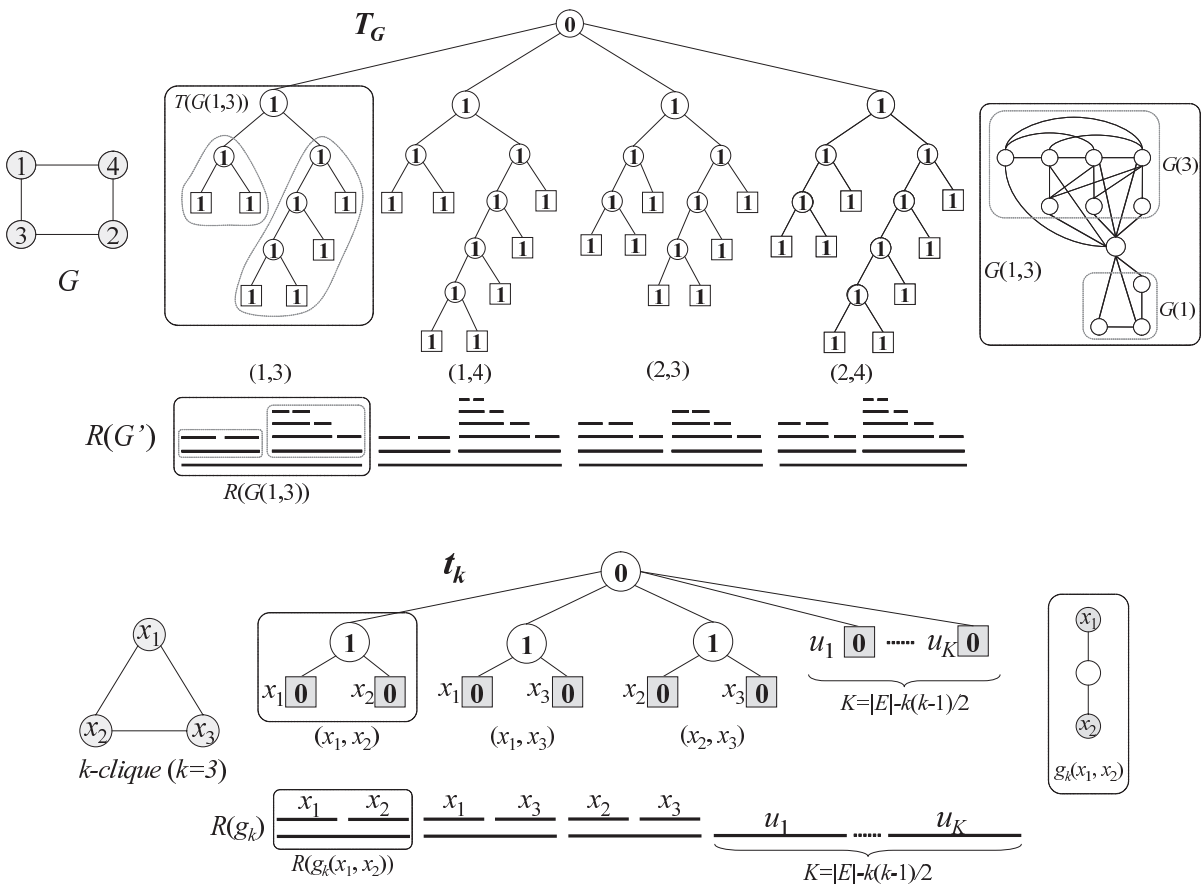

Fig. 6 A graph $G = (V, E)$ and a positive integer $k = 3$ are supposed to be instances of CLIQUE. G is transformed into an interval graph G' whose labeled PQ-tree is T_G. $G(1, 3)$ is one of the connected components of G'. And k-clique is transformed into an interval graph pattern g_k whose PQ-tree pattern is t_k. $g_k(x, y)$ is one of the connected components of g_k.

whether or not *g matches G* is NP-complete.

Proof. We reduce the CLIQUE problem, which is to decide whether a given general graph $G = (V, E)$ has a clique of size k, to the matching problem. We assume that $V = \{1, 2, \ldots, |V|\}$. We define a series of interval graphs $G(i)$ $(i = 0, 1, \ldots)$ inductively as follows. Let $G(0) = (\{b\}, \emptyset)$. For $G(i-1) = (V_{i-1}, E_{i-1})$ and two new vertices a and c $(a, c \notin V_{i-1})$ $(i \geq 1)$, we define $G(i) = (V_{i-1} \cup \{a, c\}, E_{i-1} \cup \{(a, v) \mid v \in V_{i-1}\} \cup \{(a, c)\})$. Each vertex $i \in \{1, 2, \ldots, |V|\}$ is transformed uniquely into an interval graph $G(i)$ (see Fig. 5). Let $G(i, j)$ be the interval graph constructed from $G(i)$, $G(j)$ and a new vertex by connecting the new vertex to all the vertices of $G(i)$ and $G(j)$. For example, $G(1, 3)$ is shown in Fig. 6. Although no labeled PQ-tree is needed for this reduction, for easy understanding, we draw the labeled PQ-trees of the interval graphs in Figs. 5 and 6. By using $G(i)$ and $G(i, j)$ $(1 \leq i, j \leq |V|)$, we transform the graph G to an interval graph G' whose labeled PQ-tree is T_G (see Fig. 6). The root of T_G has just $|E|$ children each of which represents an edge in G. In Fig. 6, we omit the interval graph G' because it is too complicated to draw. Instead of it, we show one of the interval representations $R(G')$ of G'. In a similar way, we transform k-clique to an interval graph g_k whose PQ-tree pattern is t_k. Let $K = |E| - k(k-1)/2$. And let $x_1, \ldots, x_k$ and

$u_1, \ldots, u_K$ be mutually distinct variable labels. The root of t_k has just $|E|$ children which represent $k(k-1)/2$ edges of k-clique and K variables for garbage collections. For $k = 3$, we show one of the interval representations $R(g_k)$ of g_k in Fig. 6.

Next we show that this reduction is computed in polynomial time with respect to the size of G and k. The root of the labeled PQ-tree T_G of G' has exactly $|E|$ children. The subtree rooted at each child represents an edge in E. For each edge $(i, j) \in E$, the subtree corresponding to (i, j) contains exactly $(2i+1)+(2j+1)+1 = 2i+2j+3$ nodes. Thus, T_G contains $1 + \sum_{(i,j)\in E}(2i + 2j + 3) \leq 1 + |E|(2(|V| - 1) + 2|V| + 3) = O(|E||V|)$ nodes. The interval graph G' also contains $O(|E||V|)$ vertices, and therefore $O(|E|^2|V|^2)$ edges. The interval graph pattern g_k reduced from k-clique has $|E|-k(k-1)$ vertices and $k(k-1)$ edges. G' and g_k are directly computed from G and k by simply replacing vertices and edges with their corresponding interval graphs and connections among the graphs. Therefore we conclude that this is a polynomial time reduction with respect to the size of G and k.

Finally we show that g_k matches G' if and only if G has a k-clique. Both the interval graph G' and the interval graph pattern g_k consist of $|E|$ connected components, each of which includes exactly one vertex which is adjacent to the other vertices in the component. Here we call the vertex the root of the component. Let $x_1, \ldots, x_k$ be the variable labels appearing in the connected components of g_k which contain more than one vertex. We assume that g_k matches G'. Then there exists a substitution τ such that $G' \equiv g_k\tau$. There is an isomorphism φ from G' to $g_k\tau$. Since the roots of the components of G' are transformed into those of g_k by φ, there is an injection f from $\{x_1, \cdots, x_k\}$ to $\{1, \cdots, |V|\}$ such that τ contains bindings $x_1/G(f(x_1)), \ldots, x_k/G(f(x_k))$. Therefore the subgraph induced by $\{f(x_1), \cdots, f(x_k)\}$ of G is a k-clique. Conversely we assume that G has a k-clique. Let $v_1, \ldots, v_k$ be the vertices by which the k-clique is induced, where $\{v_1, \ldots, v_k\} \subseteq \{1, \ldots, |V|\}$. Let $\theta = [x_1/G(v_1), \ldots, x_k/G(v_k)]$ and $\eta = [u_1/G(w_1, w'_1), \ldots, u_K/G(w_K, w'_K)]$ where $u_1, \ldots, u_K$ are the variables for garbage collections and $\{(w_1, w'_1), \ldots, (w_K, w'_K)\} = E - \{(v_i, v_j) \mid 1 \leq i < j \leq k\}$. And let $\tau = \theta\eta$. Since for any i and j $(1 \leq i < j \leq k)$, the variables labeled with x_i and x_j are connected through a vertex in g_k, and $G(v_i)$ and $G(v_j)$ are also connected through a vertex in G', we conclude that $g_k\tau$ is isomorphic to G'. □

Next we give a polynomial time algorithm for deciding, given a linear interval graph pattern $g \in \mathcal{LIGP}$ and a given interval graph $G \in \mathcal{IG}$, whether or not g matches G. From Theorem 1 and Lemma 1, we have the following lemma.

Lemma 2: For $g \in \mathcal{IGP}$ and $G \in \mathcal{IG}$, g matches G if and only if $T(g)$ matches $T(G)$

Firstly, we transform g and G into a linear PQ-tree pattern $T(g)$ and a labeled PQ-tree $T(G)$, respectively. Secondly, we decide whether or not there is a substitution τ such that $T(G) \equiv T(g)\tau$. Below, we briefly denote $T(G)$ by T and $T(g)$ by t. For a node a of a PQ-tree pattern t, we denote by $t[a]$ the PQ-tree pattern induced by a and all the nodes and variables which are descendants of a.

Definition 7: Let t and T be a PQ-tree pattern and a labeled PQ-tree, respectively. For any node $a \in V(t)$, we say that a subset of $V(T)$ is the *candidate set* of a, denoted by $NS(a)$, if it satisfies that, for any node $b \in V(T)$, $b \in NS(a)$ if and only if $t[a]$ matches $T[b]$.

For $v \in V(t) \cup V(T)$, we denote by $depth(v)$ the depth of a node v and by $ch(v)$ the number of children of a node v. The following algorithm computes $NS(a)$ for each $a \in V(t)$ by using $NS(a_1), \ldots, NS(a_{ch(a)})$ where $a_1, \ldots, a_{ch(a)}$ are all children of a. We assign a candidate set to each node of a given linear PQ-tree pattern t. The algorithm terminates when a candidate set is assigned to the root of t. From the definition of a candidate set, $NS(r_t)$ contains the root of T if and only if t matches T, where r_t is the root of t.

Since no variable of t has a child, for any substitution τ and a node $a \in V(t)$, the depth of a in $t\tau$ is equal to that of a of t. Therefore, in order to decide whether or not t matches T, we only need to compute $NS_d(a) = \{b \in NS(a) \mid depth(b) = depth(a)\}$ for all $a \in V(t)$. Below we give an algorithm for computing the sets $NS_d(a)$ for all node $a \in V(t)$. The assignment method depends on the type of a node a. The label of a node, which is a nonnegative integer or a sequence of pairs of positive integers, plays an important role for the assignment.

Leaf: $NS_d(a)$ is the set of all leaves $b \in V(T)$ such that $depth(b) = depth(a)$ and $Label(b) = Label(a)$.

Variable: $NS_d(a)$ is the set of all nodes $b \in V(T)$ such that $depth(b) = depth(a)$ and either

(1) b is a leaf and $Label(b) > Label(a)$,
(2) b is a P-node and $Label(b) \geq Label(a)$, or
(3) b is a Q-node and $Label(b)$ contains $\underbrace{(1, ch(b)), \ldots, (1, ch(b))}_{\text{at least } Label(a) \text{ times}}$.

P-node: $NS_d(a)$ is the set of all P-nodes $b \in V(T)$ satisfying the following three conditions.

(1) $depth(b) = depth(a)$ and $Label(b) = Label(a)$.
(2) If there is a child a' of a such that a' is a variable and $Label(a') = 0$, then $ch(b) \geq ch(a)$, otherwise, $ch(b) = ch(a)$.
(3) Let $a_1, \ldots, a_{ch(a)}$ and $b_1, \ldots, b_{ch(b)}$ be the children of a and b, respectively. Then there is an index subsequence $m_1, \ldots, m_{ch(a)}$ $(1 \leq m_1 < \cdots < m_{ch(a)} \leq ch(b))$ such that $b_{m_k} \in NS_d(a_k)$ for all k $(1 \leq k \leq ch(a))$.

The condition (3) is decided by computing the maximum graph matching for a bipartite graph $B = (U, V, E)$ where $U = \{a_1, \ldots, a_{ch(a)}\}$, $V = \{b_1, \ldots, b_{ch(b)}\}$, $E = \{(a_i, b_j) \mid 1 \leq i \leq ch(a),\ 1 \leq j \leq ch(b),\ b_j \in NS_d(a_i)\}$. A node b is in $NS_d(a)$ if and only if b satisfies (1), (2) and that B has the maximum matching of size $ch(a)$.

Q-node: $NS_d(a)$ is the set of all Q-nodes $b \in V(T)$ satisfy-

ing the following three conditions.

(1) $ch(b) = ch(a)$ and $depth(b) = depth(a)$.
(2) Either $Label(b) = Label(a)$ or $Label(b) = Label^r(a)$ holds.
(3) Let $a_1, \ldots, a_{ch(a)}$ and $b_1, \ldots, b_{ch(a)}$ be the ordered children of a and b, respectively. Then, for all i $(1 \leq i \leq ch(a))$, if $Label(b) = Label(a)$, then $b_i \in NS_d(a_i)$, otherwise, $b_{ch(a)-i+1} \in NS_d(a_i)$.

Lemma 3: The problem of deciding, given a linear PQ-tree pattern $t \in \mathcal{LPQTP}$ and a labeled PQ-tree $T \in \mathcal{PQT}$, whether or not t matches T is solvable in $O(nN^{1.5})$ time, where $n = |V(t)|$ and $N = |V(T)|$.

Proof. The correctness follows from the following fact. For a node $b \in V(T)$, $b \in NS_d(a)$ if and only if $t[a]$ matches $T[b]$ and $depth(b) = depth(a)$. This is shown by induction on the depth of a node $a \in V(t)$ in a bottom-up manner. We omit the proof because it is long but not difficult.

Next we analyze the time complexity of the algorithm. Let d be the height of t, and n_i and N_i $(0 \leq i \leq d)$ the numbers of nodes of depth i of t and T, respectively. For a node $a \in V(t)$ of depth i, if a is either a leaf or a variable, we need $O(N_i)$ time to compute the set $NS_d(a)$. If a is a P-node, we compute a maximum matching problem for a bipartite graph. It is known that the maximum matching problem for a given bipartite graph G can be computed in $O(|E(G)|\sqrt{|V(G)|})$ time [4]. Thus we need $O(ch(a)ch(b)\sqrt{ch(a)+ch(b)})$ time to decide whether or not a P-node $b \in V(T)$ is in $NS_d(a)$. Let $K_{i,\max} = \max\{ch(b) \mid b \text{ is a P-node of depth } i \text{ in } V(T)\}$. Then we need $O(ch(a)N_{i+1}\sqrt{K_{i,max}})$ time for computing $NS_d(a)$. In the case of a Q-node a, we need $O(N_{i+1})$ time for computing $NS_d(a)$, since we must examine all children of Q-nodes of depth i of T. Therefore we need $O(n_{i+1}N_{i+1}\sqrt{K_{i,max}})$ time to compute all nodes of depth i of t. Since a node of depth d of t is either a leaf or a variable, the total time for computing $NS_d(a)$ for all nodes $a \in V(t)$ is $O(\sum_{i=0}^{d-1} n_{i+1}N_{i+1}\sqrt{K_{i,max}} + n_dN_d)$ time. Since $\sum_{i=0}^{d} n_i = n$, $\sum_{i=0}^{d} N_i \leq N$, and $\max\{K_{i,max} \mid 0 \leq i \leq d\} \leq N$,

$$\sum_{i=0}^{d-1} n_{i+1}N_{i+1}\sqrt{K_{i,max}} \leq (\sum_{i=0}^{d} n_i)(\sum_{i=0}^{d} N_i)\sqrt{N} = nN^{1.5}.$$

Therefore we need $O(nN^{1.5})$ time to decide whether or not t matches T. □

Theorem 3: The problem of deciding, given a linear interval graph pattern $g \in \mathcal{LIGP}$ and an interval graph $G \in \mathcal{IG}$, whether or not g matches G is solvable in polynomial time.

Proof. Firstly, we transform g and G into a PQ-tree pattern $T(g)$ and a labeled PQ-tree $T(G)$, respectively. These transformations need $O(n+m)$ and $O(N+M)$ times, respectively, where $n = |V(g)| + |H(g)|$, $m = |E(g)|$, $N = |V(G)|$, and $M = |E(G)|$ [6]. Since g has at most n maximal cliques, the number of leaves of $T(g)$ is at most n. Since any internal node of $T(g)$ has at least 2 children, we have $|V(T(g))| = O(n)$. Similarly we have $|V(T(G))| = O(N)$. From Lemma 3, we need $O(nN^{1.5})$ time to decide whether or not $T(g)$ matches $T(G)$. Then the total complexity is $O(nN^{1.5} + m + M)$ time. □

4. Minimally Generalized Linear Interval Graph Patterns

For an interval graph pattern g, let $L(g) = \{G \in \mathcal{IG} \mid g \text{ matches } G\}$. For a finite set of interval graphs $S \subset \mathcal{IG}$, a *minimally generalized linear interval graph pattern* explaining S is defined as a linear interval graph pattern $g \in \mathcal{LIGP}$ such that $S \subseteq L(g)$ and there exists no linear interval graph pattern $g' \in \mathcal{LIGP}$ satisfying $S \subseteq L(g') \subsetneq L(g)$.

For a PQ-tree pattern t, the PQ-tree pattern language $L_T(t)$ of t is defined as $\{T \in \mathcal{PQT} \mid t \text{ matches } T\}$. For a finite set of PQ-trees $S_T \subset \mathcal{PQT}$, a *minimally generalized linear PQ-tree pattern* explaining S_T is defined as a linear PQ-tree pattern $t \in \mathcal{LPQTP}$ such that $S_T \subseteq L_T(t)$ and there exists no linear PQ-tree pattern $t' \in \mathcal{LPQTP}$ satisfying $S_T \subseteq L_T(t') \subsetneq L_T(t)$.

Lemma 4: For $S \subset \mathcal{IG}$ and $g \in \mathcal{LIGP}$, g is a minimally generalized linear interval graph pattern explaining S if and only if $T(g)$ is a minimally generalized linear PQ-tree pattern explaining $S_T = \{T(g) \mid g \in S\}$.

Proof. From Lemma 2, we make a point that $S \subseteq L(g)$ if and only if $S_T \subseteq L_T(T(g))$. Moreover, for two linear interval graph patterns g and g', $L(g') \subseteq L(g)$ if and only if $L_T(T(g')) \subseteq L_T(T(g))$. Therefore we have the lemma. □

For a given set $S \subset \mathcal{IG}$ of interval graphs, we find a minimally generalized linear interval graph pattern explaining S in the following way. First we transform the set S into a set of labeled PQ-trees $S_T \subset \mathcal{PQT}$. Then we find a minimally generalized linear PQ-tree pattern explaining S_T. Finally we transform the obtained minimally generalized linear PQ-tree pattern explaining S_T into the minimally generalized linear interval graph pattern explaining S.

The *size* of a PQ-tree pattern t is defined as $|V(t)| + |H(t)|$. Let S_T be a finite set of PQ-trees. We define the following 4 classes of linear PQ-tree patterns s, p, $q_{(w,Z)}$, and r (Fig. 7).

- The linear PQ-tree pattern s consists of only one variable of label 1.
- The linear PQ-tree pattern p consists of one P-node with label 0 and two variables with label 0. Both variables are the children of the P-node.
- The linear PQ-tree patterns $q_{(w,Z)}$ consist of one Q-node with label w and a series of variables or leaves with

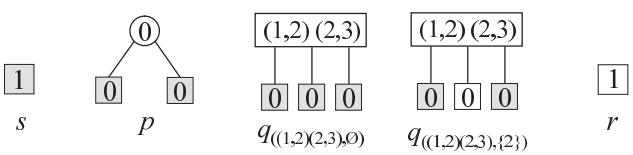

Fig. 7 A basic set of linear PQ-tree patterns. The 3rd and 4th linear PQ-tree patterns are two examples of the form $q(w, Z)$ which are obtained from the PQ-tree of minimum size in S_T.

label 0. All the variables and leaves are the children of the Q-node. The index (w, Z) satisfies the following conditions: w is the label of the Q-node which does not contain a pair $(1, m)$, where m be the number of children of the Q-node, and Z is the set $\{i \in \{1, \ldots, m\} \mid w$ contains two pairs (j, i) and (i, k), and the i-th child of the Q-node is a leaf$\}$.
From the PQ-tree of minimum size in S_T, we generate all linear PQ-tree patterns of the form $q_{(w,Z)}$ by using Procedure Extraction-Qpattern in Fig. 8.

- The linear PQ-tree pattern r consists of only one leaf node of label 1.

We give an algorithm for finding a minimally generalized PQ-tree pattern, called MINL-$\mathcal{PQTP}$.

Algorithm MINL-$\mathcal{PQTP}$

We start with the linear PQ-tree pattern t consisting of only one variable with label 0. This pattern t generates all labeled PQ-trees. Then we repeatedly apply a combination of the following refinements to all variables of a temporary linear PQ-tree pattern t while $S_T \subset L_T(t)$. Let h be a variable of t whose variable label is x.

(1) **if** $S_T \subseteq L_T(t[x/s])$ **then** $t := t[x/s]$;
(2) **if** $S_T \subseteq L_T(t[x/p])$ **then** $t := t[x/p]$;
(3) **if** there are w and Z such that $S_T \subseteq L_T(t[x/q_{(w,Z)}])$ **then** $t := t[x/q_{(w,Z)}]$;

If none of the above refinements can be applied to the current linear PQ-tree pattern t, we repeatedly apply the next refinement while $S_T \subseteq L_T(t)$.

(4) **if** $S_T \subseteq L_T(t[x/r])$ **then** $t := t[x/r]$;

If no more refinement is possible, we output the current linear PQ-tree pattern t. An example of a process of refinements is shown in Fig. 9.

Lemma 5: For a given finite set of labeled PQ-trees $S_T \subset \mathcal{PQT}$, the algorithm MINL-$\mathcal{PQTP}$ correctly outputs a minimally generalized linear PQ-tree pattern explaining S_T.

```
Procedure Extraction-Qpattern;
Input: a labeled PQ-tree T;
Output: a list of linear PQ-tree patterns;
begin
    L := an empty list;
    foreach Q-node q in T do begin
        Let ℓ be the label obtained from Label(q) by
        removing all (1, ch(q));
        q' := a Q-node with label ℓ;
        for k = 1 to ch(q) do begin
            if the k-th child of q is a leaf with label 0 then
                c := a leaf with label 0;
            else c := a variable with label 0;
            Connect c to q' as its k-th child
        end;
        Add q' to L
    end;
    return L
end
```

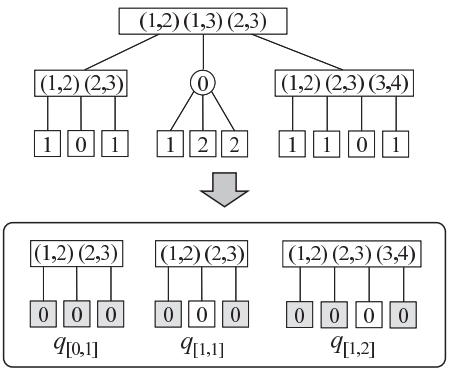

Fig. 8 Procedure Extraction-Qpattern and its example. A linear PQ-tree pattern $q_{[i,j]}$ is obtained from the j-th Q-node of depth i of the above PQ-tree by this procedure. For the above labeled PQ-tree, we have $q_{[0,1]} = q((1,2)(2,3), \emptyset)$, $q_{[1,1]} = q((1,2)(2,3), \{2\})$ and $q_{[1,2]} = q((1,2)(2,3)(3,4), \{3\})$.

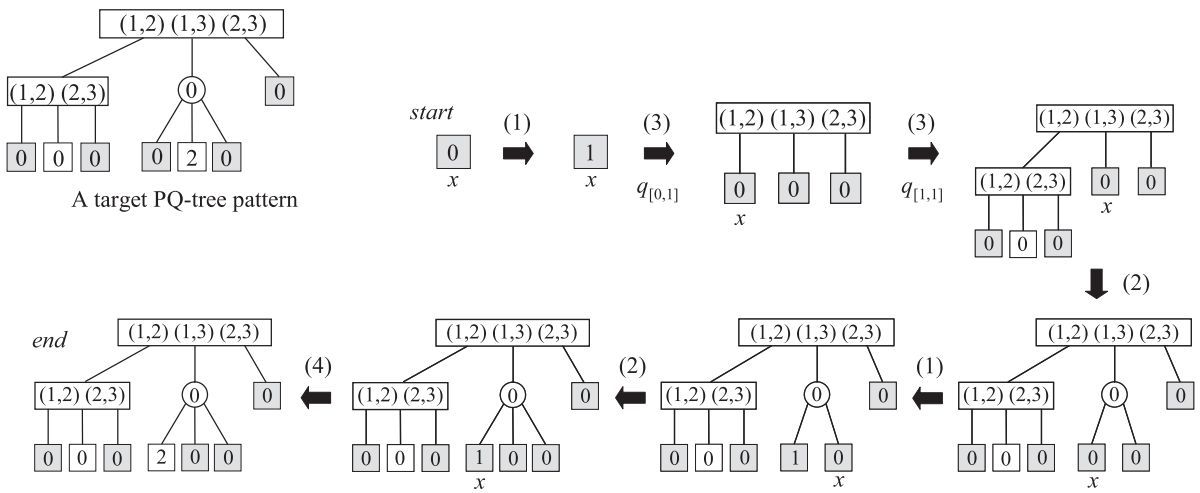

Fig. 9 A refinement process of Algorithm MINL-$\mathcal{PQTP}$. The numbers over the arrows indicate the refinements (1)–(4) used in the algorithm. The PQ-tree patterns in Fig. 8 are used for the refinements (3). The label "x" specifies the target variable of a refinement.

Proof. Let t_1 be a linear PQ-tree pattern immediately before an application of refinements (4) starts and t an output linear PQ-tree pattern of Algorithm MINL-$\mathcal{PQTP}$. We will show that if there is a linear PQ-tree pattern t' such that $S_T \subseteq L_T(t') \subseteq L_T(t)$, then $t' \equiv t$. It is easy to see that $L_T(t') \subseteq L_T(t_1)$. Let t'' be the linear PQ-tree pattern obtained by t' by replacing each leaf $v \in V(t')$ with $Label(v) \geq 1$ with a variable of label $Label(v) - 1$. Obviously $S_T \subseteq L_T(t'')$ holds. Let $x_1, \ldots, x_m$ ($m \geq 0$) be all labels of variables in t''. And let $\theta_r = [x_1/r, \ldots, x_m/r]$. Then there is a substitution θ such that $t''\theta_r \equiv t_1\theta$. Let $\theta = [y_1/s_1, \ldots, y_n/s_n]$ where $y_1, \ldots, y_n$ ($n \geq 0$) are all variables of t_1 and $s_1, \ldots, s_n$ are PQ-trees. For each s_i ($1 \leq i \leq n$), let s'_i be the linear PQ-tree pattern obtained from s_i by replacing each leaf $v \in V(s_i)$ with $Label(v) \geq 1$ with a variable of label $Label(v) - 1$. Let $\theta' = [y_1/s'_1, \ldots, y_n/s'_n]$, then we can see that $t'' \equiv t_1\theta'$ holds. If s'_i is a linear PQ-tree pattern which can be produced from a linear PQ-tree pattern in the basic set of linear PQ-tree patterns except r, this contradicts the assumption that none of refinements (1)–(3) can be applied to t_1. Therefore s'_i is the linear PQ-tree pattern consisting of only one variable of label 0. Then $t_1\theta' \equiv t_1$ holds. Consequently we can see that $t'' \equiv t_1$. Since t' can be obtained from t'' by a series of refinements (4), t' can also be obtained from t_1 by the same series of refinements (4). Let $K = \max\{Label(v) \mid v \text{ is a leaf of } t\}$. Let T be the linear PQ-tree pattern which is obtained from t' by substituting leaves of label $K + 1$ for all variables of t'. If $t' \not\equiv t$, t does not match T. This contradicts $L_T(t') \subseteq L_T(t)$. Then we have $t' \equiv t$. □

Lemma 6: For a given finite set of labeled PQ-trees $S_T \subset \mathcal{PQT}$, the algorithm MINL-$\mathcal{PQTP}$ outputs a minimally generalized linear PQ-tree pattern in $O(|S_T|N_{\min}^3 N_{\max}^{1.5})$ time, where $N_{\min}$ and $N_{\max}$ are the minimum and maximum sizes of PQ-trees in S_T.

Proof. From Lemma 3, we need $O(|S_T|N_{\min}N_{\max}^{1.5})$ time to decide whether or not $S_T \subseteq L_T(t)$. Since the refinement operations (1)–(4) are applied at most $O(N_{\min}^2)$ times, the total time complexity of this algorithm is $O(|S_T|N_{\min}^3 N_{\max}^{1.5})$ time. □

Theorem 4: For a given finite set of interval graphs $S \subset \mathcal{IG}$, a *minimally generalized linear interval graph pattern* explaining S can be computed in polynomial time.

Proof. The transformation of the set of interval graphs S to a set of labeled PQ-trees $S_T \subset \mathcal{PQT}$ needs $O(|S|(n_{\max}+m_{\max}))$ time, where $n_{\max}$ (resp. $m_{\max}$) is the maximum number of vertices (resp. edges) of interval graphs in S. From Lemma 6, we can find a minimally generalized linear PQ-tree pattern explaining S_T in $O(|S|n_{\min}^3 n_{\max}^{1.5})$ time, where $n_{\min}$ is the minimum number of vertices of interval graphs in S. Therefore the time complexity of the algorithm is $O(|S|(n_{\min}^3 n_{\max}^{1.5} + m_{\max}))$. □

5. Conclusions and Future Work

From Theorems 3 and 4, we conclude that the class of linear interval graph pattern languages is polynomial time inductively inferable from positive data by using the theorems in [1], [9]. We are now considering polynomial time learnabilities of graph languages on classes of graph structured patterns expressing probe interval graphs, chordal graphs, and outerplanar graphs, and so on.

References

[1] D. Angluin, "Finding patterns common to a set of strings," J. Comput. Syst. Sci., vol.21, pp.46–62, 1980.
[2] K.S. Booth and G.S. Lueker, "Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms," J. Comput. Syst. Sci., vol.13, pp.335–379, 1976.
[3] F. Gardi, "The mutual exclusion scheduling problem for proper interval graphs," LIF Research Report 02-2002, Laboratoire d'Informatique Fondamentale de Marseille, April 2002.
[4] J. Hopcroft and R. Karp, "An $n^{5/2}$ algorithm for maximum matching in bipartite graphs," SIAM J. Comput., vol.2, no.3, pp.225–231, 1973.
[5] T. Horváth, J. Ramon, and S. Wrobel, "Frequent subgraph mining in outerplanar graphs," Proc. KDD 2006, pp.197–206, 2006.
[6] G.S. Lueker and K.S. Booth, "A. linear time algorithm for deciding interval graph isomorphism," J. ACM, vol.26, no.2, pp.183–195, 1979.
[7] S. Matsumoto, T. Shoudai, T. Miyahara, T. Uchida, and Y. Suzuki, "Learning of finite unions of tree patterns with internal structured variables from queries," IEICE Trans. Inf. & Syst., vol.E91-D, no.2, pp.222–230, Feb. 2008.
[8] F.R. McMorris, C. Wang, and P. Zhang, "On probe interval graphs," Disc. Appl. Math., vol.88, pp.315–324, 1998.
[9] T. Shinohara, "Polynomial time inference of extended regular pattern languages," Proc. RIMS Symp. Software Science and Engineering, Springer-Verlag, LNCS 147, pp.115–127, 1982.
[10] Y. Sasaki, H. Yamasaki, T. Shoudai, and T. Uchida, "Mining of frequent block preserving outerplanar graph structured patterns," Proc. ILP 2007, Springer-Verlag, LNAI 4894, pp.239–253, 2008.
[11] Y. Suzuki, T. Shoudai, T. Miyahara, and T. Uchida, "Ordered term tree languages which are polynomial time inductively inferable from positive data," Theor. Comput. Sci., vol.350, pp.63–90, 2006.
[12] R. Takami, Y. Suzuki, T. Uchida, T. Shoudai, and Y. Nakamura, "Polynomial time inductive inference of TTSP graph languages from positive data," Proc. ILP 2005, Springer-Verlag, LNAI 3625, pp.366–383, 2005.
[13] P. Zhang, E.A. Schon, S.G. Fisher, E. Cayanis, J. Weiss, S. Kistler, and P.E. Bourne, "An algorithm based on graph theory for the assembly of contings in physical mapping of DNA," CABIOS, 10, pp.309–317, 1994.
[14] P. Zhang, "Probe interval graph and its applications to physical mapping of DNA," Int. Conf. Computational Molecular Biology RECOMB 2000 (Poster Session), 2000.

Hitoshi Yamasaki received the B.S. degree in Physics, the M.S. degree in Informatics from Kyushu University, in 2004 and 2006, respectively. He is currently a Ph.D. student in Kyushu University. His research interests include algorithmic graph theory and knowledge discovery from structured data.

Takayoshi Shoudai received the B.S. in 1986, the M.S. degree in 1988 in Mathematics and the Dr. Sci. in 1993 in Information Science all from Kyushu University. Currently, he is an associate professor of Department of Informatics, Kyushu University. His research interests include algorithmic graph theory, algorithmic learning theory, and data mining from graph-structured data. He is a member of IPSJ and ACM.