IEICE TRANS. INF. & SYST., VOL.E93–D, NO.7 JULY 2010

PAPER

Random Generation and Enumeration of Proper Interval Graphs*

Toshiki SAITOH[†a)], ***Nonmember***, **Katsuhisa YAMANAKA**[††], ***Member***, **Masashi KIYOMI**[†], ***Nonmember***, ***and*** **Ryuhei UEHARA**[†], ***Member***

SUMMARY We investigate connected proper interval graphs without vertex labels. We first give the number of connected proper interval graphs of n vertices. Using this result, a simple algorithm that generates a connected proper interval graph uniformly at random up to isomorphism is presented. Finally an enumeration algorithm of connected proper interval graphs is proposed. The algorithm is based on reverse search, and it outputs each connected proper interval graph in O(1) time.

key words: *counting, enumeration, proper interval graphs, random generation, unit interval graphs*

1. Introduction

Recently there has arisen need to process huge amounts of data in the areas of data mining, bioinformatics, etc. In order to find and classify knowledge automatically from the data, we assume that the data have a certain structure. We have to attain three efficiencies to deal with the complex structures: the structure has to be represented efficiently; essentially different instances have to be enumerated efficiently; and the properties of the structure have to be checked efficiently. In the area of graph drawing, there are several papers [5], [16], [23], [29]. From the viewpoint of graph classes, the previously studied structures are relatively primitive, and there are many unsolved problems for more complex structures: Trees are widely investigated as a model of such structured data [10], [20], [26]–[28], and recently, distance-hereditary graphs are studied [30].

A variety of graph classes have been proposed and studied [6]. Among them, interval graphs have been widely investigated since they were introduced in the 1950s by a mathematician, Hajös, and by a molecular biologist, Benzer, independently [11, Chapter 8]. A graph is called an interval graph if it represents intersecting intervals on a line. In this paper, we study a subclass of interval graphs. An interval graph is called a unit interval graph if it has a unit length interval representation. An interval representation is called proper if no interval properly contains another one on the representation. An interval graph is called a proper interval graph if it has a proper interval representation. Interestingly, unit interval graphs coincide with proper interval graphs; those notions define the same class [4], [34].

In addition to the fact that proper interval graphs form a basic graph class, they have been of interest from a viewpoint of graph algorithms. There are many problems that can be solved efficiently on proper interval graphs [3], [8], [9], [14], [15], [32]. It is also known that proper interval graphs are strongly related to the classic NP-hard problem, bandwidth problem [17]. The bandwidth problem is finding a layout of vertices; the objective is to minimize the maximum difference of two adjacent vertices on the layout. The bandwidth problem is NP-hard even on trees [25], [33]. The bandwidth problem has been studied since the 1950s; it has many applications including sparse matrix computations (see [7], [22] for survey). For any given graph $G = (V, E)$, finding a best layout of vertices is equivalent to finding a proper interval graph $G' = (V, E')$ with $E \subseteq E'$ whose maximum clique size is the minimum among all such proper interval graphs [17]. The proper interval completion problem is also motivated by molecular biology, and hence it attracts much attention (see, e.g., [18]).

In this paper, we investigate counting, random generation, and enumeration of proper interval graphs. More precisely, we aim to count, generate, and enumerate unlabeled connected proper interval graphs. We note that the graphs we deal with are unlabeled. This is reasonable to avoid redundancy from a practical point of view.

Unlabeled proper interval graphs can be naturally represented by a language over an alphabet $\Sigma = \{`[', `]'\}$. The number of strings representing proper interval graphs is strongly related to a well known notion called Dyck path, which is a staircase walk from $(0, 0)$ to (n, n) that lies strictly below (but may touch) the diagonal $x = y$. The number of Dyck paths of length n is equal to Catalan number $C(n)$. Thus, our results for counting and random generation of proper interval graphs with n vertices are strongly related to $C(n)$. The main difference is that we have to consider isomorphism and symmetry in the case of proper interval graphs. For example, to generate an unlabeled connected proper interval graph uniformly at random, we have to consider the number of valid representations of each graph since it depends on the symmetry of the graph. We show in Sect. 3 that the number of connected proper interval graphs of $n + 1$ vertices is $\frac{1}{2}(C(n) + \binom{n}{\lfloor n/2 \rfloor})$. Extending the result, we give

Manuscript received September 9, 2009.
Manuscript revised March 7, 2010.

[†]The authors are with Japan Advanced Institute of Science and Technology, Nomi-shi, 923–1292 Japan.
[††]The author is with University of Electro-Communications, Chofu-shi, 182–8585 Japan.

*A preliminary version of this article was presented at WALCOM 2009 [36].

a) E-mail: toshikis@jaist.ac.jp

DOI: 10.1587/transinf.E93.D.1816

an $O(n^3)$ time and a linear space algorithm that generates a connected proper interval graph with n vertices uniformly at random.

Our enumeration algorithm is based on the reverse search developed by Avis and Fukuda [2]. We design a good parent-child relation among the string representations of the proper interval graphs in order to perform the reverse search efficiently. The relation allows us to perform each step of the reverse search in O(1) time, and hence we have an efficient algorithm that enumerates every unlabeled connected proper interval graph with n vertices in O(1) time and $O(n)$ space. (Each graph G is output in the form of the difference of edges between G and the previous one so that the algorithm can output it in O(1) time.) Here we notice that there are some similar known algorithms that enumerate every string of '[' and ']' in constant time [20]. However, it is not always possible to obtain a constant delay algorithm for enumerating proper interval graphs from such constant delay string enumeration algorithms. For example, consider a string '[[...[]]...]' of length $2n$. This represents a complete graph of size n, and the number of edges of it is $n(n-1)/2$. If we swap the 3rd '[' and the first ']', the number of edges of the represented graph becomes $(n-1)(n-2)/2+1$ though the size of differences in the two strings is O(1). That is, an efficient enumeration algorithm for strings does not necessarily provides an efficient enumeration algorithm for graphs. Our efficient enumeration algorithm for strings also produces an efficient enumeration algorithm for graphs in a straightforward way.

We note that all the results above can be extended from "n vertices" to "at most n vertices". This will be discussed in the concluding remarks.

2. Preliminaries

A graph (V, E) with $V = \{v_1, v_2, \ldots, v_n\}$ is an *interval graph* if there is a finite set of closed intervals $\mathcal{I} = \{I_{v_1}, I_{v_2}, \ldots, I_{v_n}\}$ on the real line such that $\{v_i, v_j\} \in E$ iff $I_{v_i} \cap I_{v_j} \neq \emptyset$ for each i and j with $0 < i, j \leq n$. We call the interval set $\mathcal{I}$ an *interval representation* of the graph. For each interval I, we denote by $L(I)$ and $R(I)$ the left and right endpoints of the interval, respectively (hence we have $L(I) \leq R(I)$ and $I = [L(I), R(I)]$). For two intervals I and J, we write $I \prec J$ if $L(I) \leq L(J)$ and $R(I) \leq R(J)$.

An interval representation is *proper* if no two distinct intervals I and J exist such that I properly contains J or vice versa. That is, either $I \prec J$ or $J \prec I$ holds for every pair of intervals I and J. An interval graph is *proper* if it has a proper interval representation. If an interval graph G has an interval representation $\mathcal{I}$ such that every interval in $\mathcal{I}$ has the same length, G is said to be a *unit interval graph*. Such interval representation is called a *unit interval representation*. It is well known that proper interval graphs coincide with unit interval graphs [34]. That is, given a proper interval representation, we can transform it to a unit interval representation. A simple constructive way of the transformation can be found in [4]. With perturbations if necessary, we can assume without loss of generality that $L(I) \neq L(J)$ (and hence $R(I) \neq R(J)$), and $R(I) \neq L(J)$ for any two distinct intervals I and J in a unit interval representation $\mathcal{I}$. And we assume that the intervals in $\mathcal{I}$ are sorted by $L(I)$ values.

We denote an alphabet {'[', ']'} by Σ throughout the paper. We encode a unit interval representation $\mathcal{I}$ of a unit interval graph G by a string $s(\mathcal{I})$ in Σ^* as follows; we sweep the interval representation from left to right, and encode $L(I)$ by '[' and encode $R(I)$ by ']' for each $I \in \mathcal{I}$. We call the encoded string a *string representation* of G. We say that string x in Σ^* is *balanced* if the number of '['s in x is equal to that of ']'s. Clearly $s(\mathcal{I})$ is a balanced string of $2n$ letters. Using the construction in [4], $s(\mathcal{I})$ can be constructed from a proper interval representation $\mathcal{I}$ in $O(n)$ time and vice versa since the ith '[' and the ith ']' give the left and right endpoints of the ith interval, respectively.

We define $\overline{[}$ = ']' and $\overline{]}$ = '[' respectively. For two strings $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_m$ in Σ^*, we say that x is *smaller* than y if (1) $n < m$, or (2) $n = m$ and there exists an index $i \in \{1, \ldots, n\}$ such that $x_{i'} = y_{i'}$ for all $i' < i$ and $x_i =$ '[' and $y_i =$ ']'. If x is smaller than y, we denote $x < y$. We note that the balanced string $x = [[\cdots[]]\cdots]$ is the smallest among those of the same length. For a string $x = x_1x_2\cdots x_n$ we define the *reverse* $\bar{x}$ of x by $\bar{x} = \bar{x}_n\bar{x}_{n-1}\cdots\bar{x}_1$. A string x is *reversible* if $x = \bar{x}$. Here we have the following lemma:

Lemma 1 (See, e.g., [9, Corollary 2.5])**.** *Let G be a connected proper interval graph, and $\mathcal{I}$ and $\mathcal{I}'$ be any two unit interval representations of G. Then either $s(\mathcal{I}) = s(\mathcal{I}')$ or $s(\mathcal{I}) = \overline{s(\mathcal{I}')}$ holds. That is, the unit interval representation and hence the string representation of a proper interval graph is determined uniquely up to isomorphism.*

A connected proper interval graph G is said to be *reversible* if its string representation is reversible. Note that G is supposed to be connected in Lemma 1. If G is disconnected, we can obtain several distinct string representations by arranging the connected components.

It is easier for our purpose (counting, random generation, and enumeration of unlabeled proper interval graphs) to deal with the encoded strings in Σ^* than to use interval representations. Given an interval representation $\mathcal{I}$ of a proper interval graph G, the smaller of the two string representations $s(\mathcal{I})$ and $\overline{s(\mathcal{I})}$ is called *canonical*. If $s(\mathcal{I})$ is reversible, $s(\mathcal{I})$ is the canonical string representation. Hereafter we sometimes identify a connected proper interval graph G with its canonical string representation.

For a string $x = x_1x_2\cdots x_n \in \Sigma^n$ of length n, we define the *height* $h_x(i)$ ($i \in \{0, \ldots, n\}$) as follows;

$$h_x(i) = \begin{cases} 0 & \text{if } i = 0, \\ h_x(i-1) + 1 & \text{if } x_i = \text{'['}, \text{ and} \\ h_x(i-1) - 1 & \text{if } x_i = \text{']'}. \end{cases}$$

We say that a string x is *nonnegative* if $\min_i\{h_x(i)\}$ is equal to 0 (we do not have $\min_i\{h_x(i)\} > 0$ since $h_x(0) = 0$). The following observation is immediate:

Observation 1. *Let $x = x_1x_2\cdots x_{2n}$ be a string in Σ^{2n}. (1) x is a string representation of a (not necessarily connected)*

proper interval graph if and only if x is balanced and nonnegative. (2) x is a string representation of a connected proper interval graph if and only if x_1 = '[' and x_{2n} = ']', and the string $x_2 \cdots x_{2n-1}$ is balanced and nonnegative.

A balanced nonnegative string of length $2n$ corresponds to a well-known notion called *Dyck path*, which is a staircase walk from $(0,0)$ to (n,n) that lies strictly below (but may touch) the diagonal $x = y$. The number of Dyck paths of length n is equal to *Catalan number* $C(n) = \frac{1}{n+1}\binom{2n}{n}$; see [37, Corollary 6.2.3] for further details. We note that Observation 1 does not care about isomorphism up to reversal. We have to avoid the duplications of isomorphic graphs for counting the number of mutually nonisomorphic graphs, and for uniform random generation of them.

We define one of the generalized notions of Catalan number: The number of nonnegative strings $x = x_1 x_2 \cdots x_n$ of length n with $h_x(n) = h \geq 0$ is denoted by $C(n,h)$.

3. Counting and Random Generation

In this section, we count the number of mutually nonisomorphic proper interval graphs. We also propose an algorithm that efficiently generates a proper interval graph uniformly at random.

The number of proper interval graphs has been given by the recurrence equation in [13]. The closed equation of the number of proper interval graphs has been mentioned informally by Karttunen in 2002 [19]. We here give an explicit proof since we use some of its concepts for random generation.

Theorem 1 (Karttunen 2002)**.** *For any positive integer n, the number of connected proper interval graphs of $n+1$ vertices is $\frac{1}{2}\left(C(n) + \binom{n}{\lfloor n/2 \rfloor}\right)$.*

Proof. We define three sets $R(n), S(n)$, and $T(n)$ of strings in Σ^{2n} of length $2n$ by

$R(n) = \{x \mid x$ is balanced, nonnegative, $|x| = 2n$, and x is reversible$\}$,

$S(n) = \{x \mid x$ is balanced, nonnegative, $|x| = 2n$, and x is not reversible$\}$, and

$T(n) = \{x \mid x$ is balanced, nonnegative, and $|x| = 2n\}$.

The number of connected proper interval graphs of $n+1$ vertices is equal to $|S(n)|/2 + |R(n)| = |T(n)|/2 + |R(n)|/2 = \frac{1}{2}(C(n) + |R(n)|)$, by Observation 1. The number of elements in $R(n)$ is equal to that of nonnegative strings x' of length n, since each reversible string x is obtained by the concatenation of strings x' and $\bar{x}'$.

Now the task is the evaluation of the number of nonnegative strings x of length n with $h_x(n) = h$. Clearly we have $C(n,h) = 0$ if $h > n$. The following equations hold for each integers i and k with $0 \leq i \leq k$.

(1) $C(2k, 2i+1) = 0$, $C(2k+1, 2i) = 0$,

(2) $C(2k, 0) = C(k)$, $C(k,k) = 1$, and

(3) $C(k,i) = C(k-1, i-1) + C(k-1, i+1)$.

Let $T(k,h)$ be a set of nonnegative strings $x' = x_1 x_2 \cdots x_k$ of length k with $h_x(k) = h$, and $M(k+1, h+1)$ be $\{$'[' $x \mid x \in T(k,h)\}$. Note that $M(k+1, h+1)$ is a set of m-Raney sequences by letting $m = 2$, length $k+1$, and total sum $h+1$ in [12, Sect. 7.5], where '[' is 1 and ']' is -1. Clearly, $|T(k,h)| = |M(k+1, h+1)|$. Therefore,

$C(k,h) = |T(k,h)| = \frac{h+1}{k+1}\binom{k+1}{(k-h)/2}$

by [12, Eq. (7.69), p.349].

It is necessary to show $\sum_{i=0}^{n} C(n,i) = \binom{n}{\lfloor n/2 \rfloor}$ to complete the proof. This equation can be obtained from Eq. (5.18) in [12]. □

Next we consider the uniform random generation of a proper interval graph of n vertices.

Theorem 2. *For any given positive integer n, a connected proper interval graph with n vertices can be generated uniformly at random in $O(n)$ space. The time complexity to generate a string representation of proper interval graph uniformly is $O(n^3)$. The time complexity to convert the string representation to a graph representation is $O(n+m)$ where m is the number of edges of the created graph.*

Proof. We denote by $y = y_1 \cdots y_{2n}$ the canonical string of a connected proper interval graph $G = (V, E)$ to be obtained. We fix y_1 = '[' and y_{2n} = ']' and generate $x = x_1 \cdots x_{2n'}$ with $y = [x]$, where $n' = n - 1$ and x is a balanced nonnegative string.

The idea is simple; just generate a balanced nonnegative string x. However each non-reversible graph corresponds to two balanced nonnegative strings, while each reversible graph corresponds to exactly one balanced nonnegative (reversible) string. We use the equation $|S(n')|/2 + |R(n')| = |T(n')|/2 + |R(n')|/2 = \frac{1}{2}(C(n') + |R(n')|)$ in Theorem 1 in order to adjust the generation probabilities. The algorithm first selects which type of string to generate: (1) a balanced nonnegative string (that can be reversible) with probability $|T(n')|/(|T(n')| + |R(n')|) = C(n')/\left(C(n') + \binom{n'}{\lfloor n'/2 \rfloor}\right)$ or (2) a balanced nonnegative reversible string with probability $|R(n')|/(|T(n')| + |R(n')|) = \binom{n'}{\lfloor n'/2 \rfloor}/\left(C(n') + \binom{n'}{\lfloor n'/2 \rfloor}\right)$. This probabilistic choice adjusts the generation probabilities between reversible graphs and non-reversible graphs.

In each case, the algorithm generates each string uniformly at random using the function $C(n,h)$ introduced in the proof of Theorem 1 as follows:

Case 1: Generation of a balanced nonnegative string of length $2n'$ uniformly at random. There is a known algorithm for this purpose [1]. We simply generate sequence of '[' and ']' from left to right. Assume that the algorithm has already generated a nonnegative string $x_1 \cdots x_k$ of length k with $k < 2n'$. Next, we choose either '[' or ']' as x_{k+1}. The choice between alternative next states must be made on the basis of the proportion of terminal strings reached through the alternatives. The number of nonnegative strings that the next letter is '[' is $p = C(r, h_x(k)+1)$, and the number of nonnegative strings that the next letter is ']' is $q = C(r, h_x(k)-1)$,

where r is equal to $2n' - k - 1$. Choose '[' as the next letter with probability $\frac{p}{p+q} = \frac{(h_x(k)+2)(r-h_x(k)+1)}{2(r+1)(h_x(k)+1)}$ and choose ']' with probability $\frac{q}{p+q} = \frac{h_x(k)(r+h_x(k)+3)}{2(r+1)(h_x(k)+1)}$. Then we have a balanced nonnegative string of length $2n'$ uniformly at random.

Case 2: Generation of a balanced nonnegative reversible string of length $2n'$ uniformly at random. The desired balanced nonnegative reversible string x can be represented as $x = x_1x_2\cdots x_{n'-1}x_{n'}\bar{x}_{n'}\bar{x}_{n'-1}\cdots\bar{x}_2\bar{x}_1$, where $x_1x_2\cdots x_{n'}$ is a nonnegative string of length n'. We thus generate a nonnegative string $x' := x_1x_2\cdots x_{n'}$ of length n' uniformly at random.

Unfortunately, a similar approach to Case 1 does not work; given a positive prefix $x_1x_2\cdots x_i$, it seems to be hard to generate $x_{i+1}\cdots x_{n'}$ that ends at some $h_x(n')$ uniformly, since the string may pass below both of $h_x(i)$ and $h_x(n')$.

The key idea is to generate the desired string backwardly. This step consists of two phases. The algorithm first chooses the height $h_x(n')$ of the last letter $x_{n'}$ randomly. Then the algorithm randomly selects the height $h_x(i)$ of the ith letter x_i from $h_x(i+1)$ for each $i = n'-1, n'-2, \ldots, 1$. That is, we have either $h_x(i) := h_x(i+1) - 1$ or $h_x(i) := h_x(i+1) + 1$ in general, and $h_x(0) = 0$ at last. From the sequence of the heights, we can construct $x = x_1x_2\cdots x_{n'}$ in $O(n)$ time and space: If $h_x(i) = h_x(i+1) - 1$, we have $x_i =$ '[', and if $h_x(i) = h_x(i+1) + 1$, we have $x_i =$ ']'.

We first consider the first phase. By the proof of Theorem 1, the number of nonnegative strings ending at height h is $C(n', h)$, and $\sum_{i=0}^{n'} C(n', i) = \binom{n'}{\lfloor n'/2 \rfloor}$. Hence, for each h with $0 \le h \le n'$, the algorithm sets $h_x(n') = h$ with probability $C(n', h)/\binom{n'}{\lfloor n'/2 \rfloor}$.

Next we consider the second phase. For general i with $1 \le i < n'$, the height $h_x(i)$ is either $h_x(i) = h_x(i+1) + 1$ or $h_x(i) = h_x(i+1) - 1$. The number of nonnegative strings of length i ending at the height $h_x(i+1) + 1$ is $p = C(i, h_x(i+1) + 1)$, and the number of nonnegative strings of length i ending at the height $h_x(i+1) - 1$ is $q = C(i, h_x(i+1) - 1)$. The algorithm sets $h_x(i) = h_x(i+1) + 1$ with probability $\frac{p}{p+q} = \frac{(h_x(i+1)+2)(i-h_x(i+1)+1)}{2(i+1)(h_x(i+1)+1)}$ and sets $h_x(i) = h_x(i+1) - 1$ with probability $\frac{q}{p+q} = \frac{h_x(i+1)(i+h_x(i+1)+3)}{2(i+1)(h_x(i+1)+1)}$. The algorithm finally obtains $h_x(1) = 1$ and $h_x(0) = 0$ with probability 1 after repeating this process. The string $x' = x_1x_2\cdots x_{n'-1}x_{n'}$ can be computed from the sequence of heights by traversing the sequence of the heights backwards and hence we can obtain $x = x'\bar{x}'$.

Now we consider complexities. Binomial coefficient $\binom{n}{k}$ can be computed in $O(nk+1)$ time and $O(k+1)$ space with Iriyama's algorithm [21]. Thus Catalan number $C(n)$ can be computed in $O(n^2)$ time and in $O(n)$ space. Generalized Catalan number $C(n,k)$ can be computed in $O(n^2)$ time and in $O(n)$ space. Note that we can compute an n bit random number in $O(n)$ time provided we can compute 1 bit random number in $O(1)$ time. Since we compute the generalized Catalan number $n/2$ times in the first phase in Case 2, our random generation algorithm can be performed in $O(n^3)$ time. Note that $C(n)$ is exponentially larger than $\binom{n}{\lfloor n/2 \rfloor}$ so the probability of selecting Case 2 is close to 0. Therefore the complexity $O(n^2)$ on that depends on the selection which type of string to generate.

The generation of $x' = x_1x_2\cdots x_{n'}$ from the sequence of their heights $h_x(n'), h_x(n'-1), \ldots, h_x(1)$ in Case 2 in the proof of Theorem 2 require $O(n)$ space. Calculations of Catalan numbers also require $O(n)$ space. Hence the space complexity of the algorithm is $O(n)$. □

4. Enumeration

We here enumerate all connected proper interval graphs with n vertices. It is sufficient to enumerate each string representation of connected proper interval graphs, by Lemma 1. Let S_n be the set of balanced and canonical strings $x = x_1x_2\cdots x_{2n}$ in Σ^{2n} such that $x_1 =$ '[', $x_{2n} =$ ']', and the string $x_2\cdots x_{2n-1}$ is nonnegative. We define a tree structure, called *family tree*, in which each vertex corresponds to each string in S_n. We enumerate all the strings in S_n by traversing the family tree. Since S_n is trivial when $n = 1, 2$, we assume $n > 2$.

We start with some definitions. Let $x = x_1x_2\cdots x_{2n}$ be a string in Σ^{2n}. If $x_ix_{i+1} = [\,]$, i is called a *front index* of x. Contrary, if $x_ix_{i+1} =][$, i is called a *reverse index* of x. For example, a string [[[[ ][ ][ ]][ ]][ ][ ]] has 6 front indices 4, 6, 8, 11, 14, and 16, and has 5 reverse indices 5, 7, 10, 13, and 15. The string $[^n]^n$ in S_n is called the *root* and denoted by r_n. Let $x = x_1x_2\cdots x_{2n}$ be a string in Σ^{2n}. We denote the string $x_1x_2\cdots x_{i-1}\bar{x}_i\bar{x}_{i+1}x_{i+2}\cdots x_{2n}$ by $x[i]$ for $i = 1, 2, \ldots, 2n-1$. We define $P(x)$ by $x[j]$ for $x \in \Sigma^{2n} \setminus \{r_n\}$, where j is the minimum reverse index of x. For example, for $x =$[[[[ ][ ]][ ]][ ][ ]], we have $P(x) =$[[[[ ][ ]][ ]][ ][ ]] (the flipped pair is enclosed by the grey box and the minimum reverse indices are underlined).

Lemma 2. *For every $x \in S_n \setminus \{r_n\}$, we have $P(x) \in S_n$.*

Proof. For any $x \in S_n \setminus \{r_n\}$, it is easy to see that $P(x) = x'_1x'_2\cdots x'_{2n}$ satisfies that $x'_1 =$ '[', $x'_{2n} =$ ']', $x'_2\cdots x'_{2n-1}$ is balanced and nonnegative. Thus we show that $P(x)$ is canonical.

We first assume that x is reversible. Then the minimum reverse index j of x satisfies $j \le n$, since x is reversible and is not the root. If $j = n$, $P(x)$ is still reversible and hence $P(x)$ is canonical. When $j < n$, we have $x_{j'} = \bar{x}'_{2n-j'+1}$ for each $1 \le j' < j$, $x_j =$ '[', and $x'_{2n-j+1} =$ '['. Hence $P(x) < \overline{P(x)}$.

Next we consider the case that x is not reversible. There must be an index i such that $x_i = x_{2n-i+1} =$ '[' and $x_{i'} = \bar{x}_{2n-i'+1}$ for all $1 \le i' < i$, since x is canonical. Moreover we have $1 \le i < n$, since x is balanced. Let ℓ be the minimum reverse index of x. We first observe that $\ell \ne i$ since $x_\ell =$ ']'. We also see that $\ell < 2n - i + 1$ since $x_{i'} = \bar{x}_{2n-i'+1}$ for all $1 \le i' < i$. If $\ell < i - 1$, using a similar argument above, we have $P(x) < x < \overline{P(x)}$. If $\ell > i$, the changes to the string has

no effect; we still have x'_i = '[' and x'_{2n-i+1} = '[' and hence $P(x) < \overline{P(x)}$. The last case is $\ell = i-1$. In this case, we have $x_{i-1}x_i$ = ']', $x_{n-i+1}x_{n-i+2}$ = '[[' and $x_{i'} = \bar{x}_{2n-i'+1}$ for all $1 \le i' < i$. Thus we have $x'_{i-1}x'_i$ = '[]', $x'_{n-i+1}x'_{n-i+2}$ = '[[', and $x'_{i'} = \bar{x}'_{2n-i'+1}$ for all $1 \le i' < i-1$. This implies that $P(x) < \overline{P(x)}$. □

Next we define the family tree among strings in S_n. We call $P(x)$ *the parent* of x, and x is a *child* of $P(x)$ for each $x \in S_n \setminus \{r_n\}$. Note that $x \in S_n$ may have multiple or no children while each string $x \in S_n \setminus \{r_n\}$ has the unique parent $P(x) \in S_n$. Given a string x in $S_n \setminus \{r_n\}$, we have the unique sequence $x, P(x), P(P(x)), \ldots$ of strings in S_n by repeatedly finding the parent. We call it *the parent sequence* of x. For example, for x =[[[ ][ ]][ ]], we have $P(x)$ =[[[[ ]]][ ]], $P(P(x))$=[[[[ ]][ ]]], $P(P(P(x)))$ =[[[[ ][ ]]]], and $P(P(P(P(x))))$ = r_5. The next lemma ensures that the root r_n is the common ancestor of all the strings in S_n.

Lemma 3. *The parent sequence of x in S_n eventually ends up with r_n.*

Proof. For a string $x = x_1x_2\cdots x_{2n}$ in S_n, we define a potential function $p(x) = \sum_{i=1}^{n} 2^{n-i}b(x_i) + \sum_{i=1}^{n} 2^{i-1}(1 - b(x_{n+i}))$, where $b(\text{'['}) = 0$ and $b(\text{']'}) = 1$. For any $x \in S_n$, $p(x)$ is a non-negative integer, and $p(x) = 0$ if and only if $x = r_n$.

Suppose x is not the root r_n. Then x has the minimum reverse index, say j. If $j = n$, it is easy to see that $p(P(x)) = p(x) - 2$. We suppose that $j < n$. Then we have $p(P(x)) = p(x) - 2^{n-j} + 2^{n-j-1} = p(x) - 2^{n-j-1} < p(x)$ by the definitions of the parent and the potential function. The case $j > n$ is symmetric and we obtain $p(P(x)) < p(x)$. Therefore we eventually obtain the root r_n by repeatedly finding the parent of the derived string, which completes the proof. □

We have *the family tree* T_n of S_n by merging all the parent sequences. Each vertex in the family tree T_n corresponds to each string in S_n, and each edge corresponds to each parent-child relation. See Fig. 1 for example.

Now we give an algorithm that enumerates all the strings in S_n. The algorithm traverses the family tree by reversing the procedure of finding the parent as follows. Given a string x in S_n, we enumerate all the children of x. Every child of x is in the form $x[i]$ where i is a front index of x. We consider the following cases to find every i such that $x[i]$ is a child of x.

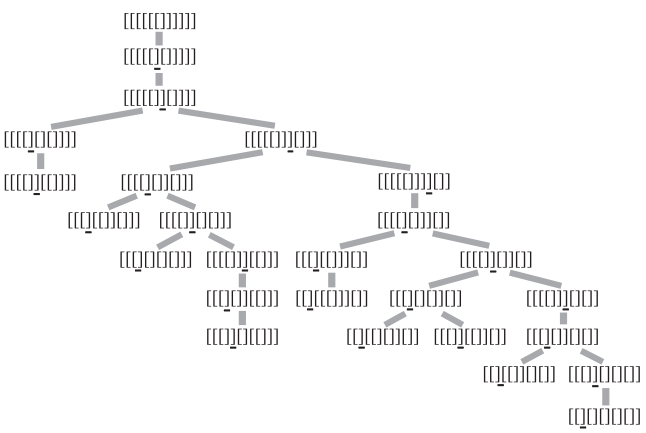

Fig. 1 The family tree T_6.

Case 1: String x is the root r_n. The string x has exactly one front index n. Since $P(x[n]) = x$, $x[n]$ is a child of x. Since $x[i]$ is not a child of x when i is not a front index, x has exactly one child.

Case 2: String x is not the root. In this case, x has at least two front indices. Let i be any front index, and j be the minimum reverse index. If $i > j+1$ then $x \neq P(x[i])$, since i is not the minimum reverse index of $x[i]$. If $i \le j+1$, $x[i]$ may be a child of x. Thus we call i satisfying the minimum front index or $j+1$ a *candidate index* of x. For a candidate index i, if $x[i]$ is in S_n (i.e. $x[i]$ is canonical), i is called a *flippable index* and $x[i]$ is a child of x. There must exist a reverse index between any two front indices. Since only the indices satisfying the minimum front index or $j+1$ can be candidate indices, x has at most two candidate indices. Thus x has at most two children. For example, x = [[[]][]] has two candidate indices 3 and 6, one reverse index 5, and one child $x[3]$ = [[][]][].

Given a string x in S_n, we can enumerate all the children of x by the case analysis above. We can traverse T_n by repeating this process from the root recursively. Thus we can enumerate all the strings in S_n.

Now we have the following algorithm and lemma.

Procedure find-all-children($x = x_1x_2\cdots x_{2n}$) // `x is the current string.`
begin
01 Output x // `Output the difference from the previous string.`
02 **for each** flippable index i
03 **find-all-children**($x[i]$) // `Case 2`
end

Algorithm find-all-strings(n)
begin
01 Output the root $x = r_n$
02 **find-all-children**($x[n]$) // `Case 1`
end

Lemma 4. *Algorithm find-all-strings(n) enumerates all the strings in S_n.*

By Lemma 4 we can enumerate all the strings in S_n. We need two more lemmas to generate each string in O(1) time. First we show an efficient construction of the candidate index list.

Lemma 5. *Given a string x in S_n and its flippable indices, we can construct the candidate index list of each child of x in $O(1)$ time.*

Proof. Let $x[i]$ be a child of x. Each string x in S_n has at most two flippable indices (see the proof of Lemma 3). Let

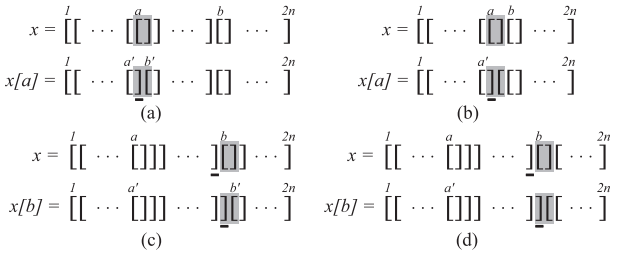

Fig. 2 Case analysis of candidate indices.

a and b be two flippable indices of x, and let a' and b' be two candidate indices of $x[a]$ or $x[b]$. We assume that $a < b$ and $a' < b'$ without loss of generality. We have the two cases below about $x[i]$.

Case 1: A child $x[a]$ of x. If $x_{a+2} =$ ']', we have two candidate indices $a' = a - 1$ and $b' = a + 1$ (see Fig. 2 (a)). Otherwise we have one candidate index $a' = a - 1$ (see Fig. 2 (b)).

Case 2: A child $x[b]$ of x. If $x_{b+2} =$ ']', we have two candidate indices $a' = a$ and $b' = b + 1$ (see Fig. 2 (c)). Otherwise we have one candidate index $a' = a$ (see Fig. 2 (d)).

By the above case analysis, a candidate index of a child either (1) appears in the previous or next index of a or b, or (2) is identical to one of x's. □

Since the number of candidate indices of x is at most two, our family tree is a binary tree. We note that a candidate index of x can become "non-candidate". In the case, such index does not become a candidate index again.

Next lemma shows that there is a method of determining whether a candidate index is flippable.

Lemma 6. *One can determine whether or not a candidate index is flippable in $O(1)$ time.*

Proof. Let $x = x_1x_2\cdots x_{2n}$ be a string in S_n and a be a candidate index of x. We denote $x[a] = y = y_1y_2\cdots y_{2n}$. A candidate index a is flippable if and only if $x[a]$ is canonical and $y_2y_3\cdots y_{2n-1}$ is nonnegative.

We first check whether or not a string $y_2y_3\cdots y_{2n-1}$ is nonnegative. We have $h_y(a) = h_x(a) - 2$ and $h_y(i) = h_x(i)$ for each $1 \le i < a$ and $a < i \le 2n$, since $a > 1$, $y_ay_{a+1} =$ '][', $x_ax_{a+1} =$ '[]', and $x_i = y_i$ for each $1 \le i < a$ and $a + 1 < i \le 2n$. Thus $y_2y_3\cdots y_{2n-1}$ is nonnegative if and only if $h_x(a) > 2$. Therefore we can check the negativity of $y_2y_3\cdots y_{2n-1}$ in O(1) time using an array of size n to maintain the sequence of heights of the string. Updates of the array also can be done in O(1) time.

We next check whether or not a string is canonical. We call $x^{\mathrm{L}} = x_1x_2\cdots x_n$ the *left string* of x, and $x^{\mathrm{R}} = \overline{x}_{2n}\overline{x}_{2n-1}\cdots\overline{x}_{n+1}$ the *right string* of x. Then x is canonical if and only if $x^{\mathrm{L}} \le x^{\mathrm{R}}$. We maintain a doubly linked list L in order to check it in O(1) time. The list L maintains the indices of different characters in x^{L} and x^{R}. First L is initialized by an empty since $x^{\mathrm{L}} = x^{\mathrm{R}}$ for $x = r_n$. In general L is empty if and only if x is reversible. We can check whether $x^{\mathrm{L}} < x^{\mathrm{R}}$ by comparing $x^{\mathrm{L}}_{L[1]}$ and $x^{\mathrm{R}}_{L[1]}$.

Now we have that x is canonical and nonnegative, $x[a]$ is nonnegative, and L consists of the different indices of x^{L} and x^{R}. Then we have $x^{\mathrm{L}}_{L[1]}$ is '[' and $x^{\mathrm{R}}_{L[1]}$ is ']'. We introduce two pointers associated to the candidate index a to update the list efficiently; two pointers p^{L}_a and p^{R}_a that point two elements in the list L. Intuitively p^{L}_a and p^{R}_a gives the two indices $L[i]$ and $L[i + 1]$ such that a is between $L[i]$ and $L[i + 1]$. When L is empty, p^{L}_a and p^{R}_a are also empty. Assume that L consists of k elements $L[1], L[2], \ldots, L[k]$. Then we have one of the following three cases. (1) If a is between $L[i]$ and $L[i + 1]$, p^{L}_a and p^{R}_a point $L[i]$ and $L[i + 1]$, respectively. More precisely, this case occurs either $1 \le a \le n$ and $L[i] \le a < L[i + 1]$ for some i or $n + 1 \le a \le 2n$ and $L[i] \le 2n - a + 1 < L[i + 1]$ for some i. (2) If a is less than $L[1]$, p^{L}_a and p^{R}_a point $L[1]$. This case occurs either $a < L[1]$ or $a > 2n - L[1]$. (3) Otherwise, i.e., the case $L[k] \le a \le 2n - L[k]$. In this case p^{L}_a and p^{R}_a point $L[k]$. Now we assume that we update x by $x[a]$, $x_ax_{a+1} =$ '[]' is replaced by $x_ax_{a+1} =$ '][']. It is straightforward and tedious that we can update the list L in O(1) time; typically, if $L[p^{\mathrm{L}}_a] < a$ and $a + 1 < L[p^{\mathrm{R}}_a]$, the algorithm inserts two new elements between p^{L}_a and p^{R}_a in L. When p^{L}_a points a, the algorithm remove it from L. The other cases are similar, and hence omitted.

The flippable index a is updated by $a - 1$ or $a + 1$ by Lemma 5. Hence the update of p^{L}_a and p^{R}_a can be done in O(1) time, which completes the proof. □

Lemmas 5 and 6 show that we can maintain the list of flippable indices of each string in O(1) time, during the traversal of the family tree. Thus we have the following lemma.

Lemma 7. *Our enumeration algorithm uses $O(n)$ space and runs in $O(|S_n|)$ time.*

By lemma 7, our algorithm generates each string in S_n in O(1) time "on average". However it may have to return from the deep recursive calls without outputting any string after generating a string corresponding to the leaf of a large subtree in the family tree. This takes much time. Therefore each string may not be generated in O(1) time in the worst case.

This delay can be canceled by outputting the strings in the "prepostorder" manner in which strings are outputted in the preorder (and postorder) manner at the vertices of odd (and even, respectively) depth of the family tree. See [31] for further details of this method; in [31] the method was not explicitly named, and the name "prepostorder" was given by Knuth [20]. Now we have the main theorem in this section.

Theorem 3. *After outputting the root in $O(n)$ time, the algorithm enumerates every string in S_n in $O(1)$ time.*

Let G and $G[i]$ be two proper interval graphs corresponding to a string x and its child $x[i]$, respectively. We note that $G[i]$ can be obtained from G by removing the one edge which represents an intersection between (1) the interval with the right endpoint corresponding to x_{i+1} and (2) one

with the left endpoint corresponding to x_i. Moreover, the root string represents a complete graph. Therefore our algorithm can be modified to deal with the graphs themselves without loss of efficiency. Note that it is not true that every constant delay enumeration algorithm for parentheses applies to that for proper interval graphs since the sizes of differences may not equal among string representations and graph representations.

Theorem 4. *After outputting the n-vertex complete graph in $O(n^2)$ time, the algorithm enumerates every connected proper interval graph of n vertices in $O(1)$ time.*

5. Conclusion

We concentrated on the graphs of n vertices in this paper. For the counting and random generation, it is straightforward to extend the results to those of graphs with at most n vertices. For the enumeration, a naive way of enumerating i-vertex proper interval graphs for each i is not sufficient, since the difference between the roots of proper interval graphs of i vertices and those of $i + 1$ vertices is not constant. However we can extend our results with suitable parent-child relation. Precisely, we extend the relation and define the parent of the root $r_i \in S_i$ by $[^i]^{i-1}[]]$ in S_{i+1}. From the viewpoint of the graphs, we add a pendant vertex to a complete graph of i vertices. From the algorithmic point of view, when the algorithm outputs the graph of $i + 1$ vertices, it recursively calls itself with the root r_i as a child of the graph, and enumerates all the smaller graphs. Thus we have the following corollary.

Corollary 1. *For any given positive integer n, (1) the number of connected proper interval graphs of at most n vertices can be computed in $O(n^3)$ time and $O(n)$ space, (2) a connected proper interval graph of at most n vertices can be generated uniformly at random, and (3) there exists an algorithm that enumerates every connected proper interval graph of at most n vertices in $O(1)$ time and $O(n)$ space.*

We investigate unlabeled connected proper interval graphs. In some cases labeled graphs may be required. Modifying our algorithms to deal with labeled graphs are straightforward. In Observation 1, it is shown that any (not necessarily connected) proper interval graph can be represented by a balanced and nonnegative string. However, in the case, we have to deal with two or more connected components. Our algorithm for enumeration can be extended to disconnected case straightforwardly, however, counting and random generation cannot be.

To deal with unlabeled graphs, it is important to determine whether or not two unlabeled graphs are isomorphic. In this sense, counting/random generation/enumeration on a graph class seems to be intractable if the isomorphism problem for the class is as hard as that for general graphs (See [38] for further details of this topic). It is known that the graph isomorphism problem can be solved in linear time for interval graphs [24]. Hence the future work would be the extensions of our algorithms to general unlabeled interval graphs.

We note that recently related results about bipartite permutation graphs are obtained [35].

References

[1] D.B. Arnold and M.R. Sleep, "Uniform random generation of balanced parenthesis strings," ACM Trans. Programming Languages and Systems, vol.2, no.1, pp.122–128, 1980.
[2] D. Avis and K. Fukuda, "Reverse search for enumeration," Discrete Appl. Math., vol.65, pp.21–46, 1996.
[3] A.A. Bertossi, "Finding hamiltonian circuits in proper interval graphs," Inf. Process. Lett., vol.17, no.2, pp.97–101, 1983.
[4] K.P. Bogart and D.B. West, "A short proof that 'proper=unit'," Discrete Mathematics, vol.201, pp.21–23, 1999.
[5] N. Bonichon, "A bijection between realizers of maximal plane graphs and pairs of non-crossing Dyck paths," Discrete Mathematics, vol.298, pp.104–114, 2005.
[6] A. Brandstädt, V.B. Le, and J.P. Spinrad, "Graph classes: A survey," SIAM, 1999.
[7] P.Z. Chinn, J. Chvátalová, A.K. Dewdney, and N.E. Gibbs, "The bandwidth problem for graphs and matrices — A survey," J. Graph Theory, vol.6, pp.223–254, 1982.
[8] C.M.H. de Figueiredo, J. Meidanis, and C.P. de Mello, "A lexBFS algorithm for proper interval graph recognition," IC DCC-04/93, Instituto de Computação, Universidade Estadual de Campinas, 1993.
[9] X. Deng, P. Hell, and J. Huang, "Linear-time representation algorithms for proper circular-arc graphs and proper Interval graphs," SIAM J. Comput., vol.25, no.2, pp.390–403, 1996.
[10] R. Geary, N. Rahman, R. Raman, and V. Raman, "A simple optimal representation for balanced parentheses," Symposium on Combinatorial Pattern Matching (CPM), Lect. Notes Comput. Sci., vol.3109, pp.159–172, Springer-Verlag, 2004.
[11] M.C. Golumbic, "Algorithmic graph theory and perfect graphs," Annals of Discrete Mathematics, vol.57, 2nd ed., Elsevier, 2004.
[12] R.L. Graham, D.E. Knuth, and O. Patashnik, Concrete Mathematics, Addison-Wesley Publishing Company, 1989.
[13] P. Hanlon, "Counting interval graphs," Trans. AMS., vol.272, no.2, pp.383–426, 1982.
[14] P. Hell and J. Huang, "Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs," SIAM J. Discrete Math., vol.18, pp.554–570, 2005.
[15] P. Hell, R. Shamir, and R. Sharan, "A fully dynamic algorithm for recognizing and representing proper interval graphs," SIAM J. Comput., vol.31, pp.289–305, 2001.
[16] Y. Kaneko and S. Nakano, "Random generation of plane graphs and its application," IEICE Trans. Fundamentals (Japanese Edition), vol.J85-A, no.9, pp.976–983, Sept. 2002.
[17] H. Kaplan and R. Shamir, "Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques," SIAM J. Comput., vol.25, no.3, pp.540–561, 1996.
[18] H. Kaplan, R. Shamir, and R.E. Tarjan, "Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs," SIAM J. Comput., vol.28, no.5, pp.1906–1922, 1999.
[19] A. Karttunen, Personal communication, 2008.
[20] D.E. Knuth, Generating All Trees, volume 4 of The Art of Computer Programming, fascicle 4 ed., Addison-Wesley, 2005.
[21] Y. Komaki and M. Arisawa, Nano Piko Kyoushitsu, Kyouritsu shuppan, 1990.
[22] Y.L. Lai and K. Williams, "A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs," J. Graph Theory, vol.31, no.2, pp.75–94, 1999.
[23] Z. Li and S.-i. Nakano, "Efficient generation of plane triangula-

tions without repetitions," International Colloquium Automata, Languages and Programming (ICALP 2001), Lect. Notes Comput. Sci., vol.2076, pp.433–443, Springer-Verlag, 2001.
[24] 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.
[25] B. Monien, "The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete," SIAM J. Alg. Disc. Meth., vol.7, no.4, pp.505–512, 1986.
[26] J.I. Munro and V. Raman, "Succinct representation of balanced parentheses, static trees and planar graphs," Proc. 38th ACM Symp. on the Theory of Computing, pp.118–126, ACM, 1997.
[27] J.I. Munro and V. Raman, "Succinct representation of balanced parentheses and static trees," SIAM J. Comput., vol.31, pp.762–776, 2001.
[28] S.-i. Nakano, "Efficient generation of plane trees," Inf. Process. Lett., vol.84, no.3, pp.167–172, 2002.
[29] S.-i. Nakano, "Enumerating Floorplans with n Rooms," IEICE Trans. Fundamentals, vol.E85-A, no.7, pp.1746–1750, July 2002.
[30] S.-i. Nakano, R. Uehara, and T. Uno, "A new approach to graph recognition and applications to distance hereditary graphs," J. Computer Science and Technology, vol.24, no.3, pp.517–533, 2009.
[31] S.-i. Nakano and T. Uno, "Constant time generation of trees with specified diameter," International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2004), Lect. Notes Comput. Sci., vol.3353, pp.33–45, Springer-Verlag, 2004.
[32] B.S. Panda and S.K. Das, "A linear time recognition algorithm for proper interval graphs," Inf. Process. Lett., vol.87, pp.153–161, 2003.
[33] C.H. Papadimitriou, "The NP-completeness of the bandwidth minimization problem," Computing, vol.16, pp.263–270, 1976.
[34] F.S. Roberts, "Indifference graphs," in Proof Techniques in Graph Theory, ed. F. Harary, pp.139–146, Academic Press, 1969.
[35] T. Saitoh, Y. Otachi, K. Yamanaka, and R. Uehara, "Random generation and enumeration of bipartite permutation graphs," 20th International Symposium on Algorithms and Computation (ISAAC 2009), Lect. Notes Comput. Sci., vol.5878, pp.1104–1113, Springer-Verlag, 2009.
[36] T. Saitoh, K. Yamanaka, M. Kiyomi, and R. Uehara, "Random generation and enumeration of proper interval graphs," Algorithms and Computation (WALCOM 2009), Lect. Notes Comput. Sci., vol.5431, pp.177–189, Springer-Verlag, 2009.
[37] R.P. Stanley, Enumerative Combinatorics, vol.2, Cambridge, 1997.
[38] R. Uehara, S. Toda, and T. Nagoya, "Graph isomorphism completeness for chordal bipartite graphs and strongly chordal graphs," Discrete Applied Mathematics, vol.145, no.3, pp.479–482, 2004.

Toshiki Saitoh received a B.S. degree from Shimane University in 2005, and M.S. degree from Japan Advanced Institute of Science and Technology in 2007. Currently, he is a student of doctor course in Information Science, Japan Advanced Institute of Science and Technology. He will receive Ph.D. degrees from Japan Advanced Institute of Science and Technology in March, 2010, and participate in the ERATO project in April, 2010.

Katsuhisa Yamanaka is an assistant professor of Graduate School of Information Systems, The University of Electro-Communications. He received B.E., M.E. and Ph.D. degrees from Gunmma University in 2003, 2005 and 2007, respectively. His research interests include combinatorial algorithms and graph algorithms.

Masashi Kiyomi was born in 1976. 2006, Doctor of Philosophy,National Institute of Informatics (in Japan). 2006–, assistant professor at School of Information Science, Japan Advanced Institute of Science and Technology.

Ryuhei Uehara received B.E., M.E., and Ph.D. degrees from the University of Electro-Communications, Japan, in 1989, 1991, and 1998, respectively. He was a researcher in CANON Inc. during 1991–1993. In 1993, he joined Tokyo Woman's Christian University as an assistant professor. He was a lecturer during 1998–2001, and an associate professor during 2001–2004 at Komazawa University. He moved to Japan Advanced Institute of Science and Technology (JAIST) in 2004, and he is now an associate professor in School of Information Science. His research interests include computational complexity, algorithms, and data structures, especially, randomized algorithms, approximation algorithms, graph algorithms, and algorithmic graph theory. He is a member of EATCS, ACM, and IEEE.