Efficient Non-isomorphic Graph Enumeration Algorithms for Subclasses of Perfect Graphs

Jun Kawahara[1], Toshiki Saitoh[2(✉)], Hirokazu Takeda[2], Ryo Yoshinaka[3], and Yui Yoshioka[2]

[1] Kyoto University, Kyoto, Japan
[2] Kyushu Institute of Technology, Kitakyushu, Japan
toshikis@ai.kyutech.ac.jp
[3] Tohoku University, Sendai, Japan

Abstract. Intersection graphs are well-studied in the area of graph algorithms. Some intersection graph classes are known to have algorithms enumerating all unlabeled graphs by reverse search. Since these algorithms output graphs one by one and the numbers of graphs in these classes are vast, they work only for a small number of vertices. Binary decision diagrams (BDDs) are compact data structures for various types of data and useful for solving optimization and enumeration problems. This study proposes enumeration algorithms for five intersection graph classes, which admit $O(n)$-bit string representations for their member graphs. Our algorithm for each class enumerates all unlabeled graphs with n vertices over BDDs representing the binary strings in time polynomial in n. Moreover, our algorithms are extended to enumerate those with constraints on the maximum (bi)clique size and/or the number of edges.

Keywords: Enumeration · Binary decision diagrams · Graph isomorphism · Graph classes · String representation

1 Introduction

This paper is concerned with efficient enumeration of unlabeled intersection graphs. An intersection graph has a geometric representation such that each vertex of the graph corresponds to a geometric object and the intersection of two objects represents an edge between the two vertices in the graph. Intersection graphs are well-studied for their practical and theoretical applications [2,17]. For example, interval graphs, which are represented by intervals on a real line, are applied in bioinformatics, scheduling, and so on [5]. Proper interval graphs are a subclass of interval graphs with interval representations where no interval is properly contained to another. These graph classes are related to important graph parameters: The bandwidth of a graph G is equal to the smallest value of the maximum clique sizes in proper interval graphs that extend G [6].

The literature has considered the enumeration problems for many of the intersection graph classes. The graph enumeration problem is to enumerate all

C.-C. Lin et al. (Eds.): WALCOM 2023, LNCS 13973, pp. 151–163, 2023.
https://doi.org/10.1007/978-3-031-27051-2_14

the graphs with n vertices in a specified graph class. If it requires not enumerating two isomorphic graphs, it is called *unlabeled*. Otherwise, it is called *labeled*. Unlabeled enumeration algorithms based on reverse search [1] have been proposed for subclasses of interval graphs and permutation graphs [14,15,18,19]. Those algorithms generate graphs in time polynomial in the number of vertices per graph. In this regard, those algorithms are considered to be fast in theory. However, since those algorithms output graphs one by one and the numbers of graphs in these classes are vast, the total running time will be impractically long, and storing the output graphs requires a large amount of space.

The idea of using *binary decision diagrams (BDDs)* has been studied to overcome the difficulty of the high complexity of enumeration. BDDs can be seen as indexing and compressed data structures for various types of data, including graphs, via reasonable encodings. The technique so-called *frontier-based search*, given an arbitrary graph, efficiently constructs a BDD which represents all subgraphs satisfying a specific property [7,10,16]. Among those, Kawahara et al. [8] proposed enumeration algorithms for several sorts of intersection graphs, e.g., chordal and interval graphs. Using the obtained BDD, one can easily count the number of those graphs, generate a graph uniformly at random, and find an optimal one under some measurement, like the minimum weight. However, the enumeration by those algorithms is labeled. In other words, the obtained BDDs by those algorithms may have many isomorphic graphs. Hence, the technique cannot be used, for example, for generating a graph at uniformly random when taking isomorphism into account.

This paper proposes polynomial-time algorithms for unlabeled intersection graph enumeration using BDDs. The five intersection graph classes in concern are those of proper interval, cochain, bipartite permutation, (bipartite) chain, and threshold graphs. It is known that the unlabeled graphs with n vertices of these classes have natural $O(n)$-bit string encodings: We require $2n$ bits for proper interval and bipartite permutation graphs [14,15] and n bits for chain, cochain, and threshold graphs [11,13]. It may be a natural idea for enumerating those graphs to construct a BDD that represents those encoding strings. Here, we remark that there are different strings that represent isomorphic graphs, and we need to keep only a "canonical" one among those strings. Actually, if we make a BDD naively represent those canonical strings, the resultant BDD will be exponentially large. To solve the problem, we introduce new string encodings of intersection graphs of the respective classes so that the sizes of the BDDs representing canonical strings are polynomial in n. Our encodings are still natural enough to extend the enumeration technique to more elaborate tasks: namely, enumerating graphs with bounded maximum (bi)clique size and/or with maximum number of edges. One application of enumerating proper interval graphs with maximum clique size k is, for example, to enumerate graphs with the bandwidth at most k. Recall that the bandwidth of a graph is the minimum size of the maximum cliques in the proper interval graphs obtained by adding edges. Thus, conversely, we can obtain graphs of bandwidth at most k by removing edges from the enumerated graphs.

2 Preliminary

Graphs. Let $G = (V, E)$ be a simple graph with n vertices and m edges. A sequence $P = (v_1, v_2, \ldots, v_k)$ of vertices is a *path from* v_1 *to* v_k if v_i and v_j are distinct for $i \neq j$ and $(v_i, v_{i+1}) \in E$ for $i \in \{1, \ldots, k-1\}$. The graph G is *connected* if for every two vertices $v_i, v_j \in V$, there exists a path from v_i to v_j. The neighbor set of a vertex v is denoted by $N(v)$, and the closed neighbor set of v is denoted by $N[v] = N(v) \cup \{v\}$. A vertex v is *universal* if $|N(v)| = n - 1$ and a vertex v is *isolate* if $|N(v)| = 0$. For $V' \subseteq V$ and $E' \subseteq E$ such that the endpoints of every edge in E' are in V', $G' = (V', E')$ is a *subgraph* of G. The graph G is *complete* if every vertex is universal. If a subgraph $G' = (V', E')$ of G is a complete graph, V' is called a *clique* of G. A clique C is *maximum* if for any clique C' in G, $|C| \geq |C'|$. A vertex set S is called an *independent* set if for each $v \in S$, $N(v) \cap S = \emptyset$. The *complement* of $G = (V, E)$ is the graph $\overline{G} = (V, \overline{E})$ where $\overline{E} = \{(u, v) \mid (u, v) \notin E\}$.

For a graph $G = (V, E)$, let (X, Y) be a partition of V; that is, $V = X \cup Y$ and $X \cap Y = \emptyset$. A graph $G = (X \cup Y, E)$ is *bipartite* if for every edge $(u, v) \in E$, either $u \in X$ and $v \in Y$ or $u \in Y$ and $v \in X$ holds. The bipartite graph G is *complete bipartite* if $E = \{(x, y) \mid x \in X, y \in Y\}$. For a subgraph $G' = (X' \cup Y', E')$ of G, $X' \cup Y'$ is called *biclique* if G' is complete bipartite. A biclique B is *maximum* if for any biclique B' in G, $|B| \geq |B'|$. Note that we here say that a biclique has the "maximum" size if the number of not edges but vertices of it is maximum. For a bipartite graph $G = (X \cup Y, E)$, $\overline{G}$ is called *cobipartite*. Note that X and Y are cliques in $\overline{G}$. An ordering $x_1, x_2, \ldots, x_{|X|}$ on X is an *inclusion ordering* if $N(x_i) \cap Y \subseteq N(x_j) \cap Y$ for every i, j with $i < j$.

Binary Strings. We use the binary alphabet $\Sigma = \{\mathtt{L}, \mathtt{R}\}$ in this paper. Let $s = c_1 c_2 \ldots c_n$ be a binary string on Σ^*. The length of s is n and we denote it by $|s|$. Let $\overline{\mathtt{L}} = \mathtt{R}$ and $\overline{\mathtt{R}} = \mathtt{L}$. For a string $s = c_1 c_2 \ldots c_n$, we define $\overline{s} = \overline{c_n}\ \overline{c_{n-1}} \ldots \overline{c_1}$. The *height* $h_s(i)$ of s at $i \in \{0, 1, \ldots, n\}$ is defined by $h_s(i) = |c_1 \ldots c_i|_{\mathtt{L}} - |c_1 \ldots c_i|_{\mathtt{R}}$, where $|t|_c$ denotes the number of occurrences of c in a string t. The string s is *balanced* if $h_s(n) = 0$; that is, the number of $\mathtt{L}$ is equal to that of $\mathtt{R}$ in s. The *height* of s is the maximum value in the height function for s and denoted by $h(s)$; that is, $h(s) = \max_i h_s(i)$. We say s is *larger* than a string s' with length n if there exists an index $i \in \{1, \ldots, n\}$ such that $h_s(i') = h_{s'}(i')$ for any $i' < i$ and $h_s(i) > h_{s'}(i)$, and we denote it by $s > s'$. The *alternate* string $\alpha(s)$ of s is obtained by reordering the characters of s from outside to center, alternately; that is, $\alpha(s) = c_1 c_n c_2 c_{n-1} \ldots c_{\lceil n/2 \rceil}$ if n is odd and $\alpha(s) = c_1 c_n c_2 c_{n-1} \ldots c_{n/2} c_{n/2+1}$ otherwise.

Binary Decision Diagrams. A *binary decision diagram (BDD)* is an edge labeled directed acyclic graph $D = (N, A)$ that classifies strings over a binary alphabet Σ of a fixed length n. To distinguish BDDs from the graphs we enumerate, we call elements of N *nodes* and those of A *arcs*. The nodes are partitioned into $n + 1$ groups: $N = N_1 \cup \cdots \cup N_{n+1}$. Nodes in N_i are said to be at *level* i for $1 \leq i \leq n + 1$. There is just one node at level 1, called the *root*.

Level $(n+1)$ nodes are only two: the 0-terminal node and the 1-terminal node. Each node in N_i for $i \leq n$ has two outgoing arcs pointing at nodes in $N_{i+1} \cup N_{n+1}$. Thus, the length of every path from the root to a node in N_i is just $i-1$ for $i \leq n$. The terminal nodes have no outgoing arcs. The two arcs from a node have different labels from Σ. We call those arcs L-arc and R-arc. When a string $s = c_1 \ldots c_n$ is given, we follow the arcs labeled $c_1, \ldots, c_n$ from the root node. If we reach the 1-terminal, then the input is accepted. If we reach the 0-terminal, it is rejected. One may reach a terminal node before reading the whole string. In that case, we do not care the rest unread suffix of the string, and classify the whole string in accordance with the terminal node. Figure 1 shows an example BDD, where LRLR and LLRR are accepted and LLRL and RLRL are rejected.

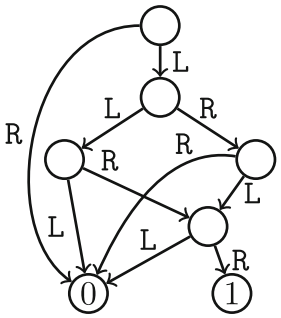

Fig. 1. An example BDD.

3 Algorithms

3.1 Proper Interval Graphs and Cochain Graphs

Definition and Properties of Proper Interval Graphs. A graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ is an *interval* graph if there exists a set of n intervals $\mathcal{I} = \{I_1, \ldots, I_n\}$ such that $(v_i, v_j) \in E$ iff $I_i \cap I_j \neq \emptyset$ for $i, j \in \{1, \ldots, n\}$. The set $\mathcal{I}$ of intervals is called an *interval representation* of G. For an interval I, we denote the left and right endpoints of I by $l(I)$ and $r(I)$, respectively. Without loss of generality, we assume that any two endpoints in $\mathcal{I}$ are distinct. An interval representation $\mathcal{I}$ is *proper* if there are no two distinct intervals I_i and I_j in $\mathcal{I}$ such that $l(I_i) < l(I_j) < r(I_j) < r(I_i)$ or $l(I_j) < l(I_i) < r(I_i) < r(I_j)$. A graph G is *proper interval* if it has a proper interval representation (Fig. 2).

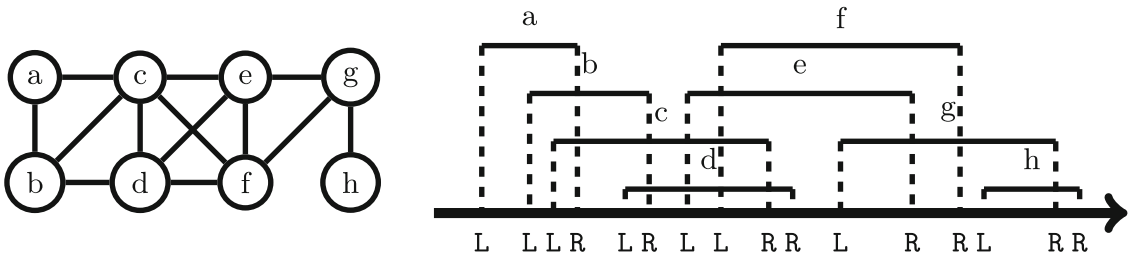

Fig. 2. Proper interval graph and its proper interval representation. The string representation of the proper interval representation is LLLRLRLLRRLRRLRR.

Proper interval graphs can be represented by binary strings as follows. Let G be a proper interval graph with n vertices and $\mathcal{I}$ be a proper interval representation of G. We can represent $\mathcal{I}$ as a string by sweeping $\mathcal{I}$ from left to right and

encoding $l(I)$ by L and $r(I)$ by R, respectively. We denote the obtained string by $s(\mathcal{I})$ and call it the *string representation* of $\mathcal{I}$. The length of $s(\mathcal{I})$ is $2n$.

Lemma 1 ([15]). *Let $s(\mathcal{I}) = c_1c_2 \ldots c_{2n}$ be a string representation of a connected proper interval graph G with n vertices.*

1. $c_1 = \mathtt{L}$ *and* $c_{2n} = \mathtt{R}$,
2. $s(\mathcal{I})$ *is balanced; that is, the number of* L *is same as that of* R *in* $s(\mathcal{I})$, *and*
3. $h_{s(\mathcal{I})}(i) > 0$ *for* $i \in \{1, \ldots, 2n-1\}$.

A connected proper interval graph has at most two string representations [4]. More strictly, for any two string representations s and s' of a connected proper interval graph G, $s = s'$ or $s = \overline{s'}$. The string representation is said to be *canonical* if $s > \overline{s}$ or $s = \overline{s}$. Thus, the canonical string representations have one-to-one correspondence to the proper interval graphs up to isomorphism [15].

Algorithm for n Vertices. We here present an enumeration algorithm of all connected proper interval graphs with n vertices up to isomorphism. We would like to construct a BDD representing all canonical string representations of proper interval graphs. However, for the efficiency of the BDD construction as described later, we instead construct a BDD representing alternate strings of all canonical representations of proper interval graphs.

We describe an overview of our algorithm. We construct the BDD in a breadth-first manner in the direction from the root node to the terminals. We create the root node in N_1, and for each node in N_i ($i \in \{1, \ldots, 2n\}$), we create its L and R-arcs and make each arc point at one of the existing nodes in N_{i+1} or N_{2n+1} or a newly created node. We call making an arc point at 0-terminal node *pruning*. For each node ν, we store into ν information on the paths from the root to ν as a tuple, which we call *state*. Two nodes having the same state never exist. When creating an (L or R) arc of a node, we compute the state of the destination from the state of the original node. If there is an existing node having the same state as the computed one, we make the arc point at the existing node, which we call *(node) sharing*.

Consider deciding whether a string s in Σ^{2n} is canonical or not; that is, $s > \overline{s}$ or $s = \overline{s}$ holds. Suppose that $s = c_1c_2 \ldots c_{2n}$ and we have $\overline{s} = \overline{c_{2n}}\,\overline{c_{2n-1}} \ldots \overline{c_1}$. This can be done by comparing c_i with $\overline{c_{2n-i+1}}$ for $i = 1, \ldots, 2n$. When creating a node ν in the BDD construction process, we would like to conduct pruning early if we can determine that all the path labels from the root via ν will not be canonical. That is the reason we adopt alternate string representations. A node in level i ($\in \{1, \ldots, 2n\}$) corresponds to the $\lceil i/2 \rceil$th character in the string representation if i is odd, and the $(2n + 1 - i/2)$th one otherwise. For example, consider the path LRLRRR. Any path extending LRLRRR will represent a string of the form $s = \mathtt{LLR}t\mathtt{RRR}$ for which $\overline{s} = \mathtt{LLL}\bar{t}\mathtt{LRR}$ for some $t \in \Sigma^*$ and $s < \overline{s}$ holds. This implies s cannot be canonical. The path goes to the 0-terminal.

We make each node, say ν, maintain state $(i, h_{\mathrm{L}}, h_{\mathrm{R}}, F)$. The first element i is the level where ν is. We take an arbitrary path from the root node to ν, say $c_1c_{2n}c_2c_{2n-1} \ldots c_{\lceil i/2 \rceil - 1}c_{2n+2-\lceil i/2 \rceil}$ (the case where i is odd) or $c_1c_{2n}c_2c_{2n-1} \ldots c_{2n+2-\lceil i/2 \rceil}c_{\lceil i/2 \rceil}$ (the case where i is even). The second and third elements $h_{\mathrm{L}}, h_{\mathrm{R}}$

represent the heights of the sequences $c_1c_2\dots c_{\lfloor i/2\rfloor}$ and $\overline{c_{2n}}\,\overline{c_{2n-1}}\dots\overline{c_{2n+2-\lceil i/2\rceil}}$, respectively. Note that we must design an algorithm so that it is well-defined; that is, the values of the sequences obtained from all the paths from the root node to ν are the same. F represents whether $(\star)$ $c_{\hat{\imath}} = \overline{c_{2n+1-\hat{\imath}}}$ holds for all $\hat{\imath} = 1,\dots,\lceil i/2\rceil - 1$. If $F = \top$, $(\star)$ does not hold; that is, there exists i' such that $c_{i'} \neq \overline{c_{2n+1-i'}}$. If $c_{i'} = \mathtt{R}$ and $\overline{c_{2n+1-i'}} = \mathtt{L}$, the canonicity condition does not meet. As shown later, such a node never exists because we conduct the pruning. Therefore, $F = \top$ means that $c_{i'} = \mathtt{L}$, $\overline{c_{2n+1-i'}} = \mathtt{R}$ and $c_{i''} = \overline{c_{2n+1-i''}}$ holds for all $i'' \leq i' - 1$, which implies that the canonicity condition is satisfied whatever the other characters are. $F = \bot$ means that $(\star)$ holds.

We discuss how to store states and conduct pruning in the process of the BDD construction. We make the root node have the state $(1, 0, 0, \bot)$. Let ν be a node that has the state $(i, h_{\mathrm{L}}, h_{\mathrm{R}}, F)$ and $\nu_{\mathtt{L}}$ and $\nu_{\mathtt{R}}$ be nodes pointed at by $\mathtt{L}$-arc and $\mathtt{R}$-arc of ν. If $i = 1$, $\nu_{\mathtt{R}}$ is 0-terminal, and if $i = 2$, $\nu_{\mathtt{L}}$ is 0-terminal because of the condition (i) in Lemma 1. First, we consider the case where i is odd. $\mathtt{L}$-arc and $\mathtt{R}$-arc of ν mean that the $\lceil i/2\rceil$th character is $\mathtt{L}$ and $\mathtt{R}$, respectively. We make $\nu_{\mathtt{L}}$ have state $(i + 1, h_{\mathrm{L}} + 1, h_{\mathrm{R}}, F)$. As for $\mathtt{R}$-arc, if $h_{\mathrm{L}} - 1 \leq 0$, we make $\mathtt{R}$-arc of ν point at 0-terminal because $\mathtt{R}$-arc means $c_{\lceil i/2\rceil} = \mathtt{R}$ and the height of $c_1c_2\dots c_{\lfloor (i+1)/2\rfloor}$ violates the condition of (iii) in Lemma 1. Otherwise, we make $\nu_{\mathtt{R}}$ have state $(i + 1, h_{\mathrm{L}} - 1, h_{\mathrm{R}}, F)$. Next, we consider the case where i is even. $\mathtt{L}$-arc and $\mathtt{R}$-arc of ν mean that the $(2n + 1 - \lceil i/2\rceil)$th character is $\mathtt{L}$ and $\mathtt{R}$, respectively. If $F = \top$, we make $\nu_{\mathtt{L}}$ and $\nu_{\mathtt{R}}$ maintain states $(i + 1, h_{\mathrm{L}}, h_{\mathrm{R}} - 1, \top)$ and $(i + 1, h_{\mathrm{L}}, h_{\mathrm{R}} + 1, \top)$, respectively. (Recall that since $F = \top$ means that the canonicity condition has already been satisfied, we need not update F.) We conduct pruning for $\mathtt{L}$-arc if $h_{\mathrm{R}} - 1 \leq 0$. Let us consider the case where $F = \bot$. Recall that $(\star)$ holds. Although we want to compare the $\lceil i/2\rceil$th and $(2n+1-\lceil i/2\rceil)$th characters to decide whether the canonicity condition holds or not, ν does not have the information on the $\lceil i/2\rceil$th character. Instead, ν has h_{L} and h_{R}. We consider two cases (i) and (ii): (i) If $h_{\mathrm{L}} - 1 = h_{\mathrm{R}}$, it means that the $\lceil i/2\rceil$th character is $\mathtt{L}$. In this case, $\mathtt{R}$-arc of ν means that the $(2n+1-\lceil i/2\rceil)$th character is $\mathtt{R}$, which implies that $(\star)$ still holds. Therefore, we make $\nu_{\mathtt{R}}$ maintain state $(i+1, h_{\mathrm{L}}, h_{\mathrm{R}}+1, \bot)$. $\mathtt{L}$-arc of ν means that the $(2n+1-\lceil i/2\rceil)$th character is $\mathtt{L}$, which implies that $(\star)$ no longer holds and the canonicity condition is satisfied. Therefore, we make $\nu_{\mathtt{L}}$ maintain state $(i + 1, h_{\mathrm{L}}, h_{\mathrm{R}} - 1, \top)$. (ii) If $h_{\mathrm{L}} - 1 \neq h_{\mathrm{R}}$, it means that the $\lceil i/2\rceil$th character is $\mathtt{R}$. In this case, $\mathtt{R}$-arc of ν means that the $(2n + 1 - \lceil i/2\rceil)$th character is $\mathtt{R}$, which violates the canonicity condition. We make $\mathtt{R}$-arc of ν point at 0-terminal. $\mathtt{L}$-arc of ν means that the $(2n+1-\lceil i/2\rceil)$th character is $\mathtt{L}$, which implies that $(\star)$ still holds. Therefore, we make $\nu_{\mathtt{L}}$ maintain state $(i + 1, h_{\mathrm{L}}, h_{\mathrm{R}} - 1, \bot)$.

Consider the case where $i = 2n$ (final level). Let the computed state as the destination of $\mathtt{L}$- or $\mathtt{R}$-arc of a node in N_{2n} be $(2n + 1, h'_{\mathrm{L}}, h'_{\mathrm{R}}, F')$. If $h'_{\mathrm{L}} \neq h'_{\mathrm{R}}$, the destination is pruned (0-terminal) because it violates the condition of (ii) in Lemma 1. Otherwise, we make the arc point at 1-terminal.

Theorem 1. *Our algorithm constructs a BDD representing all canonical string representations of connected proper interval graphs in* $\mathrm{O}(n^3)$ *time and space.*

Proof. We here analyze the complexity of the algorithm. For each level $i \in \{0, 1, \ldots, 2n\}$, the number of nodes in N_i is $\mathrm{O}(n^2)$ because $0 \leq h_{\mathrm{L}}, h_{\mathrm{R}} \leq n$ and $F \in \{\bot, \top\}$. Thus, the total size of BDD is $\mathrm{O}(n^3)$. The computation of the next state for each node can be run in constant time because it has only increment and we can access the nodes in constant time by using $\mathrm{O}(n^2)$ pointers. □

Algorithm for Maximum Clique Size k. We here present an algorithm that given natural numbers n and k, enumerates all proper interval graphs with n vertices and the maximum clique size at most k. It is well known that a clique of an interval graph G corresponds to overlap intervals of a point in an interval representation of G [3]. The number of overlapping intervals is same as the height of string representation of a proper interval graph. Thus, the enumeration of all proper interval graphs with the maximum clique size at most k can be seen as that of all canonical string representations with the height at most k. We modify the algorithm for n vertices by adding one pruning for the case when either of the heights h_{L} or h_{R} becomes larger than k. Therefore, our extended algorithm runs in $\mathrm{O}(k^2 n)$ time and space since the ranges of h_{L} and h_{R} become k from n.

Algorithm for m Edges. To extend the algorithm for n vertices and m edges, we here show how to count the number of edges from the string representation. Let s be a string representation of a proper interval graph with m edges. Sweeping the string representation from left to right, for each $i \in \{1, \ldots, 2n\}$ with $c_i = \mathtt{L}$, the height $h_s(i)$ is the number of intervals I_j with $j < i$ that overlap with i. This means that the vertex v corresponding to c_i is incident to $h_s(i)$ edges in G. Thus, we obtain the number of edges from the string representation as follows.

Lemma 2. *Let $s = c_1 \ldots c_{2n}$ be a string representation of a connected proper interval graph G with m edges and J be the set of indices i of s such that $c_i = \mathtt{L}$. The summation of heights in J is equal to m; i.e., $\sum_{i \in J} h_s(i) = m$.*

In the construction of a BDD, each node stores the value to maintain the number of edges m'. The state of each node is now a quintuple $(i, h_{\mathrm{L}}, h_{\mathrm{R}}, F, m')$. For the L-arc of a node ν, the number of edges m' is updated to $m' + h_{\mathrm{L}}$ if i is odd and to $m' + h_{\mathrm{R}} - 1$ otherwise. When either i is odd and $m' + h_{\mathrm{L}} > m$ or i is even and $m' + h_{\mathrm{R}} - 1 > m$ holds, we make the L-arc of ν point at the 0-terminal since the number of edges is larger than m. We make each arc point at the 1-terminal if it gives a state $(2n + 1, h, h, F, m)$ for some h and F based on the state updating rule. Otherwise, it must point at the 0-terminal. For each $i \in \{1, \ldots, 2n\}$, the number of nodes in N_i is $\mathrm{O}(n^2 m)$ since $0 \leq h_{\mathrm{L}}, h_{\mathrm{R}} \leq n$ and $0 \leq m' \leq m$ and the number of levels is $2n$. Therefore, the algorithm runs in $\mathrm{O}(n^3 m)$ time.

Theorem 2. *A BDD representing all connected proper interval graphs with n vertices and maximum clique size k and with n vertices and m edges can be constructed in $\mathrm{O}(k^2 n)$ time and $\mathrm{O}(n^3 m)$ time, respectively.*

Cochain Graphs. A graph $G = (X \cup Y, E)$ is a *cochain* graph if G is cobipartite and each of X and Y has an inclusion ordering. In other words, X

and Y are cliques in G and we have two orderings over $X = \{x_1, \ldots, x_{n_X}\}$ and $Y = \{y_1, \ldots, y_{n_Y}\}$ such that $(x_i, y_j) \in E$ implies $(x_{i'}, y_{j'}) \in E$ for any $i \leq i'$ and $j \leq j'$. It is well-known [2] that cochain graphs are a subclass of proper interval graphs. Here, we give a concrete proper interval representation $\{I_1, \ldots, I_{n_X}, J_1, \ldots, J_{n_Y}\}$ of G, where x_i and y_j correspond to I_i and J_j, respectively, by

- $l(I_1) < \cdots < l(I_{n_X}) < r(I_1) < \cdots < r(I_{n_X}) < r(J_{n_Y})$,
- $l(I_{n_X}) < l(J_{n_Y}) < \cdots < l(J_1) < r(J_{n_Y}) < \cdots < r(J_1)$,
- $l(J_j) < r(I_i)$ iff $(x_i, y_j) \in E$ for $1 \leq i \leq n_X$ and $1 \leq j \leq n_Y$.

The inclusion ordering constraint guarantees that the above is well-defined and gives a proper interval representation. Therefore, one can specify a cochain graph as a proper interval graph by a $2n$-bit string representation. Moreover, the strong restriction of cochain graphs allows us to reduce the number of bits to specify a cochain graph. Obviously, the first n_X bits of the proper interval string representation of a cochain graph are all `L` and the last n_Y bits are all `R`. Thus, those $n = n_X + n_Y$ bits are redundant and removable. Indeed, one can recover the numbers n_X and n_Y from the remaining n bits. Since every surviving bit of `R` corresponds to $r(I_i)$ for some i, the number of those bits is just n_X. Similarly, n_Y is the number of bits of `L` in the new n-bit representation. Conversely, every n-bit string s can be seen as the string representation of a cochain graph with n vertices. However, the n-bit strings are not in one-to-one correspondence to the cochain graphs because universal vertices in the cochain graphs can be seen in either X or Y. To avoid the duplication, we assume that all universal vertices are in Y, so we only consider n-bit strings without `R` as a suffix. Using this n-bit string representation, we obtain an enumeration algorithm for cochain graph, and it runs in $\mathrm{O}(n)$ time.

For the constraint problems, we use $2n$-bit strings because we need to compute the size of cliques or the number of edges. Our algorithms with constraints for cochain graphs are similar to that of proper interval graphs and need to recognize whether the strings represent cochain graphs.

Theorem 3. *A BDD representing all canonical string representations of cochain graphs with n vertices, n vertices and maximum clique size k, and n vertices and m edges can be constructed in $\mathrm{O}(n)$, $\mathrm{O}(k^2 n)$, and $\mathrm{O}(n^3 m)$ time, respectively.*

3.2 Bipartite Permutation Graphs and Chain Graphs

Definition and Properties of Bipartite Permutation Graphs. Let π be a permutation on V; that is, π is a bijection from V to $\{1, \ldots, n\}$. We define $\overline{\pi}$ as $\overline{\pi}(v) = n + 1 - \pi(v)$ for all $v \in V$. We denote by π^{-1} the inverse of π.

A graph $G = (V, E)$ is *permutation* if it has a pair (π_1, π_2) of two permutations on V such that there exists an edge $(u, v) \in E$ iff $(\pi_1(u) - \pi_1(v))(\pi_2(u) - \pi_2(v)) < 0$. The pair $\mathcal{P} = (\pi_1, \pi_2)$ can be seen as the following intersection model

on two parallel horizontal lines L_1 and L_2: the vertices in V are arranged on the line L_1 (resp. line L_2) according to π_1 (resp. π_2). Each vertex w corresponds to a line segment l_w, which joins w on L_1 and w on L_2. An edge (u, v) is in E iff l_u and l_v intersects, which is equivalent to $(\pi_1(u) - \pi_1(v))(\pi_2(u) - \pi_2(v)) < 0$. The model $\mathcal{P} = (\pi_1, \pi_2)$ is called a *permutation diagram*. A graph G is *bipartite permutation* if G is bipartite and permutation.

Let $\mathcal{P} = (\pi_1, \pi_2)$ be a permutation diagram of a connected bipartite permutation graph $G = (V, E)$. Let us observe properties of π_1 and π_2, which are discussed in [14]. First, there is no vertex $u \in V$ such that $\pi_1(u) = \pi_2(u)$ unless $n = 1$. Secondly, for all vertices $u, v \in V$ such that $\pi_1(u) < \pi_2(u)$, $\pi_1(v) < \pi_2(v)$ and $\pi_1(u) < \pi_1(v)$ hold, $\pi_2(u) > \pi_2(v)$ does not hold; that is, l_u and l_v never intersects. Therefore, $X = \{u \mid \pi_1(u) < \pi_2(u)\}$ and $Y = \{u \mid \pi_1(u) > \pi_2(u)\}$ give the vertex partition of G. By expressing the above observation with the intersection model, the line segments are never straight vertical and classified into X and Y depending on their tilt directions: lines in X go from upper left to lower right and those in Y go from lower left to upper right.

Based on the above discussion, let us give a string representation $s(\mathcal{P})$ of the permutation diagram $\mathcal{P}$. We define $s_x(\mathcal{P}) = x_1 \dots x_n$ and $s_y(\mathcal{P}) = y_1 \dots y_n$ as follows: For $i = 1, \dots, n$, $x_i = \mathtt{L}$ if $\pi_1(\pi_1^{-1}(i))(= i) < \pi_2(\pi_1^{-1}(i))$, and $x_i = \mathtt{R}$ otherwise. Similarly, for $i = 1, \dots, n$, $y_i = \mathtt{R}$ if $\pi_2(\pi_2^{-1}(i))(= i) > \pi_1(\pi_2^{-1}(i))$, and $y_i = \mathtt{L}$ otherwise. In other words, $x_i = \mathtt{L}$ iff the ith intersection point of L_1 is with a line segment from X in the intersection model. On the other hand, $y_i = \mathtt{L}$ iff the ith intersection point of L_2 is with a line segment from Y. We define the string representation $s(\mathcal{P})$ of $\mathcal{P}$ by $s(\mathcal{P}) = x_1 y_1 x_2 y_2 \dots x_n y_n$. The string representation $s(\mathcal{P})$ has the following properties [14].

Lemma 3. *Let $s = c_1 c_2 \dots c_{2n}$ be a string representation of a connected bipartite permutation graph G with n vertices. Then,*

(i) $c_1 = \mathtt{L}$ *and* $c_{2n} = \mathtt{R}$,
(ii) s *is balanced; that is, the number of* $\mathtt{L}$ *is the same as that of* $\mathtt{R}$ *in* s, *and*
(iii) $h_s(i) > 0$ *for* $i \in \{1, \dots, 2n-1\}$.

By horizontally, vertically, and rotationally flipping $\mathcal{P}$, we obtain essentially equivalent diagrams $\mathcal{P}^{\mathrm{V}} = (\pi_2, \pi_1)$, $\mathcal{P}^{\mathrm{H}} = (\overline{\pi_1}, \overline{\pi_2})$, and $\mathcal{P}^{\mathrm{R}} = (\overline{\pi_2}, \overline{\pi_1})$ of G, respectively.

Lemma 4 ([14]). *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be permutation diagrams of a connected bipartite permutation graph. At least one of the equations $s(\mathcal{P}_1) = s(\mathcal{P}_2)$, $s(\mathcal{P}_1) = s(\mathcal{P}_2^{\mathrm{V}})$, $s(\mathcal{P}_1) = s(\mathcal{P}_2^{\mathrm{H}})$, or $s(\mathcal{P}_1) = s(\mathcal{P}_2^{\mathrm{R}})$ holds.*

A string representation $s(\mathcal{P})$ is said to be *canonical* if all the inequalities $s(\mathcal{P}) \geq s(\mathcal{P}^{\mathrm{V}})$, $s(\mathcal{P}) \geq s(\mathcal{P}^{\mathrm{H}})$, and $s(\mathcal{P}) \geq s(\mathcal{P}^{\mathrm{R}})$ hold.

Algorithm for n Vertices. We construct the BDD representing the set of bipartite permutation graphs using the alternate strings of the canonical representation strings. Each BDD node is identified with a state tuple $(i, h_{\mathrm{L}}, h_{\mathrm{R}}, c_{\mathrm{L}}, c_{\mathrm{R}}, F_{\mathrm{V}}, F_{\mathrm{H}}, F_{\mathrm{R}})$. The integer i is the level where the node is. The

heights h_{L} and h_{R} are those of $x_1x_2\dots x_n$ and $\overline{y_n}\,\overline{y_{n-1}}\dots\overline{y_1}$, respectively, the purpose of which is the same as in Sect. 3.1.

Let us describe $F_{\mathrm{V}}, F_{\mathrm{H}}$ and F_{R}. F_{R} is $\bot$ or $\top$, which is used for deciding whether $s(\mathcal{P}) \geq s(\mathcal{P}^{\mathrm{R}})$ holds or not. Recall that if $s(\mathcal{P}) = x_1y_1x_2y_2\dots x_ny_n$, $s(\mathcal{P}^{\mathrm{R}}) = \overline{y_n}\,\overline{x_n}\,\overline{y_{n-1}}\,\overline{x_{n-1}}\dots\overline{y_1}\,\overline{x_1}$. According to the variable order $\alpha(s(\mathcal{P}))$, we can decide whether $s(\mathcal{P}) > s(\mathcal{P}^{\mathrm{R}})$ holds or not using the heights h_{L} and h_{R} by the way described in Sect. 3.1. Then, F_{R} has the same role as F in Sect. 3.1. Next, we consider F_{V}, which is used for deciding the canonicity of $s(\mathcal{P}) \geq s(\mathcal{P}^{\mathrm{V}})$. Recall that if $s(\mathcal{P}) = x_1y_1x_2y_2\dots x_ny_n$, $s(\mathcal{P}^{\mathrm{V}}) = y_1x_1y_2x_2\dots y_nx_n$. We need to compare x_1 with y_1, y_1 with $x_1, \dots$, and y_n with x_n in order. Recall that on the BDD, the value of y_i is represented by arcs of each node in level $4i-1$. The value of x_i has already been determined by arcs of a node in level $4i-3$. Therefore, to compare x_i with y_i, we store the value of x_i into nodes. Strictly speaking, if i is odd, then, $c_{\mathrm{L}} = x_{\lceil i/2\rceil-1}$ and $c_{\mathrm{R}} = y_{2n-\lceil i/2\rceil+2}$. If i is even, then, $c_{\mathrm{L}} = x_{i/2}$ and $c_{\mathrm{R}} = y_{2n-i/2+2}$. The stored values c_{L} and c_{R} are also used for deciding whether $s(\mathcal{P}) \geq s(\mathcal{P}^{\mathrm{H}})$ holds or not in a similar way.

We estimate the number of BDD nodes by counting the possible values of a state $(i, h_{\mathrm{L}}, h_{\mathrm{R}}, c_{\mathrm{L}}, c_{\mathrm{R}}, F_{\mathrm{V}}, F_{\mathrm{H}}, F_{\mathrm{R}})$. Since $1 \leq i \leq 2n$, $0 \leq h_{\mathrm{L}} \leq n$, $0 \leq h_{\mathrm{R}} \leq n$, and the number of possible states of $c_{\mathrm{L}}, c_{\mathrm{R}}, F_{\mathrm{V}}, F_{\mathrm{H}}, F_{\mathrm{R}}$ are two, the number of possible values of tuples is $2n \times (n+1)^2 \times 2^5 = \mathrm{O}(n^3)$.

Algorithm for m edges. We present an algorithm that constructs the BDD representing the set of (string representations of) bipartite permutation graphs with n vertices and m edges when n and m are given. The number of edges of a bipartite permutation graph G is that of intersections of the permutation diagram of G. We use the following lemma.

Lemma 5. *The number of edges is* $\sum_{i=1}^{n} h_{s(\mathcal{P})}(2i)$.

We can easily obtain

$$\sum_{i=1}^{n} h_{s(\mathcal{P})}(2i) = \sum_{i=1}^{\lceil n/2\rceil} h_{s(\mathcal{P})}(2i) + \sum_{i=1}^{\lfloor n/2\rfloor} h_{s(\mathcal{P}^{\mathrm{R}})}(2i). \tag{1}$$

To count the number of edges, we store this value into each BDD node. Let us describe the detail. We make each BDD node maintain a tuple $(i, h_{\mathrm{L}}, h_{\mathrm{R}}, c_{\mathrm{L}}, c_{\mathrm{R}}, F_{\mathrm{V}}, F_{\mathrm{H}}, F_{\mathrm{R}}, m')$. The first eight elements are the same as the ones described above. The last element m' is the current value of (1). Thus, the running time of the algorithm is $\mathrm{O}(n^3m)$.

Theorem 4. *A BDD representing all connected bipartite permutation graphs with n vertices, and n vertices and m edges can be constructed in* $\mathrm{O}(n^3)$ *and* $\mathrm{O}(n^3m)$ *time, respectively.*

Chain Graphs. A graph $G = (X \cup Y, E)$ is a *chain* graph if G is bipartite and each of X and Y has an inclusion ordering. Let $(x_1, x_2, \dots, x_{|X|})$ and $(y_1, y_2, \dots, y_{|Y|})$ be an inclusion ordering of X and Y, respectively. Chain graphs are known to be a subclass of bipartite permutation graphs [2] and have the following permutation diagrams $\mathcal{P} = (\pi_1, \pi_2)$ [12]:

– $\pi_1 = (x_1, x_2, \ldots, x_{|X|}, y_{|Y|}, y_{|Y|-1}, \ldots, y_1)$,
– for $i, j \in \{1, \ldots, |X|\}$ with $i < j$, $\pi_2(x_i) < \pi_2(x_j)$,
– for $i, j \in \{1, \ldots, |Y|\}$ with $i < j$, $\pi_2(y_j) < \pi_2(y_i)$.

Chain graphs as bipartite permutation graphs have $2n$-bit string representations based on the permutation diagrams. From the diagram and Lemma 4, we observe that the string of π_1 is uniquely determined except for exchanging X and Y. Since π_1 can be fixed as above, any chain graph can be represented using an n-bit string by sweeping π_2: The ith element of π_2 is encoded as `L` if $\pi_2^{-1}(i) \in X$ and is encoded as `R` if $\pi_2^{-1}(i) \in Y$. If a chain graph G is disconnected, G consists of two parts: a connected chain graph component and a set of isolated vertices [9]. We observe that the connected chain graphs have a one-to-one correspondence with the string representations up to reversal [13]. On the other hand, isolated vertices may arbitrarily belong to X or Y. To determine a unique string representation, we assume that isolated vertices are all in X, where the representation strings must not end with `R`. Thus, we obtain an algorithm to construct a BDD representing all canonical n-bit string representations of chain graphs and it runs in $O(n)$. For the restriction problems, we adopt $2n$-bit strings defined as representations of bipartite permutation graphs instead of n-bit representations to compute the number of edges or the size of bicliques. In the algorithms, we need to check whether the constructed strings represent chain graphs satisfying the conditions described above.

Theorem 5. *A BDD representing all chain graphs with n vertices, n vertices and maximum biclique size k, and n vertices and m edges can be constructed in $O(n)$, $O(k^2 n)$, $O(n^3 m)$ time, respectively.*

3.3 Threshold Graphs

A graph G is a *threshold* graph if the vertex set of G can be partitioned into X and Y such that X is a clique and Y is an independent set and each of X and Y has an inclusion ordering. Threshold graphs are a subclass of interval graphs, and any threshold graph can be constructed by the following process [2,11]. First, if the size of the vertex set is one, the graph is threshold. Then, for a threshold graph G, (1) the graph by adding an isolated vertex to G is also threshold, and (2) the graph adding a universal vertex to G is also threshold. The sequence of the two operations (1) and (2) to construct a threshold graph is called a *construction sequence*. It is easy to see that the two threshold graphs G_1 and G_2 are not isomorphic if the construction sequences of (1) and (2) of G_1 and G_2 are different. From this characterization of threshold graphs, we obtain algorithms to construct a BDD representing all unlabeled threshold graphs by encoding the construction sequences of the operation (1) to `L` and (2) to `R`.

Theorem 6. *A BDD representing all threshold graphs with n vertices, n vertices and maximum clique size k, and n vertices and m edges can be constructed in $O(n)$ time, $O(kn)$ time, and $O(nm)$ time, respectively.*

Acknowledgments. The authors are grateful for the helpful discussions of this work with Ryuhei Uehara. This work was supported in part by JSPS KAKENHI Grant Numbers JP18H04091, JP19K12098, JP20H05794, and JP21H05857.

References

1. Avis, D., Fukuda, K.: Reverse search for enumeration. Discret. Appl. Math. **65**(1–3), 21–46 (1996)
2. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. Society for Industrial and Applied Mathematics (1999)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)
4. Deng, X., Hell, P., Huang, J.: Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. SIAM J. Comput. **25**(2), 390–403 (1996)
5. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, vol. 57). Elsevier (2004)
6. Kaplan, H., Shamir, R.: Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques. SIAM J. Comput. **25**(3), 540–561 (1996)
7. Kawahara, J., Inoue, T., Iwashita, H., Minato, S.: Frontier-based search for enumerating all constrained subgraphs with compressed representation. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **100**(9), 1773–1784 (2017)
8. Kawahara, J., Saitoh, T., Suzuki, H., Yoshinaka, R.: Colorful frontier-based search: implicit enumeration of chordal and interval subgraphs. In: Kotsireas, I., Pardalos, P., Parsopoulos, K.E., Souravlias, D., Tsokas, A. (eds.) SEA 2019. LNCS, vol. 11544, pp. 125–141. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34029-2_9
9. Kijima, S., Otachi, Y., Saitoh, T., Uno, T.: Subgraph isomorphism in graph classes. Discret. Math. **312**(21), 3164–3173 (2012)
10. Knuth, D.: The Art of Computer Programming, Volume 4A: Combinatorial Algorithms. No. Part 1, Pearson Education, London (2014)
11. Mahadev, N., Peled, U.: Threshold Graphs and Related Topics. Elsevier, Amsterdam (1995)
12. Okamoto, Y., Uehara, R., Uno, T.: Counting the number of matchings in chordal and chordal bipartite graph classes. In: Paul, C., Habib, M. (eds.) WG 2009. LNCS, vol. 5911, pp. 296–307. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11409-0_26
13. Peled, U.N., Sun, F.: Enumeration of difference graphs. Discret. Appl. Math. **60**(1–3), 311–318 (1995)
14. Saitoh, T., Otachi, Y., Yamanaka, K., Uehara, R.: Random generation and enumeration of bipartite permutation graphs. J. Discrete Algorithms **10**, 84–97 (2012)
15. Saitoh, T., Yamanaka, K., Kiyomi, M., Uehara, R.: Random generation and enumeration of proper interval graphs. IEICE Trans. Inf. Syst. **93**(7), 1816–1823 (2010)
16. Sekine, K., Imai, H., Tani, S.: Computing the Tutte polynomial of a graph of moderate size. In: Staples, J., Eades, P., Katoh, N., Moffat, A. (eds.) ISAAC 1995. LNCS, vol. 1004, pp. 224–233. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0015427
17. Spinrad, J.P.: Efficient Graph Representations. American Mathematical Society, Providence, RI, Fields Institute monographs (2003)

18. Yamazaki, K., Qian, M., Uehara, R.: Efficient enumeration of non-isomorphic distance-hereditary graphs and ptolemaic graphs. In: Uehara, R., Hong, S.-H., Nandy, S.C. (eds.) WALCOM 2021. LNCS, vol. 12635, pp. 284–295. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68211-8_23
19. Yamazaki, K., Saitoh, T., Kiyomi, M., Uehara, R.: Enumeration of nonisomorphic interval graphs and nonisomorphic permutation graphs. Theor. Comput. Sci. **806**, 310–322 (2020)