Nakano S-i, Uehara R, Uno T. A new approach to graph recognition and applications to distance-hereditary graphs. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 24(3): 517–533 May 2009

A New Approach to Graph Recognition and Applications to Distance-Hereditary Graphs*

Shin-ichi Nakano[1], *Member*, *ACM*, *IEEE*, Ryuhei Uehara[2], *Member*, *ACM*, *IEEE*, and Takeaki Uno[3]

[1] *Department of Computer Science, Faculty of Engineering, Gunma University, Gunma 376-8515, Japan*

[2] *School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan*

[3] *National Institute of Informatics, Tokyo 101-8430, Japan*

E-mail: nakano@cs.gunma-u.ac.jp; uehara@jaist.ac.jp; uno@nii.jp

Received March 10, 2008; revised February 26, 2009.

Abstract Algorithms used in data mining and bioinformatics have to deal with huge amount of data efficiently. In many applications, the data are supposed to have explicit or implicit structures. To develop efficient algorithms for such data, we have to propose possible structure models and test if the models are feasible. Hence, it is important to make a compact model for structured data, and enumerate all instances efficiently. There are few graph classes besides trees that can be used for a model. In this paper, we investigate distance-hereditary graphs. This class of graphs consists of isometric graphs and hence contains trees and cographs. First, a canonical and compact tree representation of the class is proposed. The tree representation can be constructed in linear time by using prefix trees. Usually, prefix trees are used to maintain a set of strings. In our algorithm, the prefix trees are used to maintain the neighborhood of vertices, which is a new approach unlike the lexicographically breadth-first search used in other studies. Based on the canonical tree representation, efficient algorithms for the distance-hereditary graphs are proposed, including linear time algorithms for graph recognition and graph isomorphism and an efficient enumeration algorithm. An efficient coding for the tree representation is also presented; it requires $\lceil 3.59n \rceil$ bits for a distance-hereditary graph of n vertices and $3n$ bits for a cograph. The results of coding improve previously known upper bounds (both are $2^{O(n \log n)}$) of the number of distance-hereditary graphs and cographs to $2^{\lceil 3.59n \rceil}$ and 2^{3n}, respectively.

Keywords algorithmic graph theory, cograph, distance-hereditary graph, prefix tree, tree representation

1 Introduction

Data-driven computations are of interest for data mining, bioinformatics, etc. Such computations process huge amounts of data; they find and classify knowledge automatically. For these purposes, we sometimes assume that the data is structured, and such structures can be observed implicitly or explicitly. More precisely, we assume a possible structure for the data, and then enumerate them and test if the assumption is feasible. The frequent pattern discovery problem in data mining is a typical example, and it has been widely investigated (e.g., [1–4]). Once a feasible model is found, we solve the problem at hand for the structured data. However, these structures are relatively primitive from the graph algorithmic point of view, and there are many unsolved problems for more complex structures.

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. Except for studies on trees[5−9], there have been few studies from the viewpoint of efficiency.

A variety of graph classes have been proposed and studied[10,11]. Since the early work by Rose, Tarjan, and Lueker[12], the lexicographic breadth first search (LexBFS) has played an important role as a basic tool to recognize several graph classes. The LexBFS gives us a simple ordering of vertices based on the neighborhood-preferred manner (see [13] for further details).

We use prefix trees instead of LexBFS. A prefix tree, which is also known as a trie[14], represents a set of strings and enable one to check whether a given string is included in the set or not. We regard a string as a set of neighbors of a vertex and maintain the strings of all vertices with a prefix tree. By using two prefix trees

Regular Paper

*The extended abstract was presented at the 4th Annual Conference on Theory and Applications of Models of Computation (TAMC07). This work was partially done while the second and third authors were visiting ETH Zürich, Switzerland.

corresponding to two different neighborhoods, we can efficiently find a pair of vertices having identical (or similar) neighbors. This is a different approach from the previously known algorithms based on LexBFS[15,16].

We apply the above idea to distance-hereditary graphs. Distance in graphs is one of the most important topics in algorithmic graph theory, and there are many applications that have geometric properties that can be represented by graphs. Distance-hereditary graphs were characterized by Howorka[17] as a means of dealing with the geometric distance property called *isometric*, which means that all induced paths between pairs of vertices have the same length. More precisely, a distance-hereditary graph is a graph in which the distance between any pair of vertices u and v will be the same on any vertex induced connected subgraph containing u and v. Intuitively, any longer path between them has a shortcut on any vertex induced subgraph. (Without loss of generality, we will assume that the distance-hereditary graph is connected.)

There has been some research on characterizing distance-hereditary graphs[18−20]. In particular, Bandelt and Mulder[18] showed that any distance-hereditary graph can be obtained from K_2 by a sequence of extensions called "adding a pendant vertex" and "splitting a vertex." Many efficient algorithms on distance-hereditary graphs are based on this characterization[21−26]. We will show that the extensions can be efficiently found on prefix trees that represent two different neighborhoods of vertices.

The class of distance-hereditary graphs contains two well-known graph classes: trees and cographs. Cographs can be obtained by a sequence of "splitting a vertex" extensions from K_2 and an efficient tree representation, called a cotree, is known (see [16] for further details). Recently, distance-hereditary graphs attract attention again; Oum showed that a graph is distance-hereditary if and only if it has rank-width at most 1 in his thesis[27]; Chandler *et al.* investigate probe versions of distance-hereditary graphs and cographs[28], and Gioan and Paul give a characterization by using split decomposition with a simple recognition algorithm[29].

Our study presented here makes two key contributions to the topic of distance-hereditary graphs. First, we propose a compact and canonical tree representation. This is a natural generalization of a cotree, which is the tree representation of cographs. Secondly, we show a linear time and space algorithm that constructs the tree representation for any distance-hereditary graph. The linear time and space algorithm involves two key ideas. The first idea is using two prefix trees, one for open and the other for closed neighborhoods. This idea allows us to efficiently find "twins", which are obtained by splitting a vertex. The second idea is introducing a vertex ordering named "levelwise laminar ordering", which allows us to remove pendant vertices efficiently. The levelwise laminar ordering is weaker than the LexBFS ordering. These results have the following applications.

1) The graph isomorphism problem can be solved in linear time and space. This was conjectured by Spinrad in [30, p.309], but it was not explicitly shown.

2) The recognition problem can be solved in linear time and space. Our algorithm is much simpler than the previously known recognition algorithm for the class (see [15, Chapter 4]); the original Hammer and Maffray's algorithm[20] fails in some cases, and Damiand, Habib, and Paul show a correct algorithm in [15]. However, the correct algorithm requires one to build the cotree of a cograph in linear time as a subroutine. The subroutine can be implemented by using a classic algorithm due to Corneil, Perl, and Stewart[31], or a recent algorithm based on the multisweep LexBFS approach by Bretscher, Corneil, Habib, and Paul[16]. (Note that the recent result by Gioan and Paul also gives another linear time recognition algorithm for the class[29].)

3) Given n, all distance-hereditary graphs with at most n vertices can be enumerated in $O(n)$ time per graph with $O(n^2)$ space.

4) We propose an efficient encoding of a distance-hereditary graph. Any distance-hereditary graph with n vertices can be represented in at most $\lceil 3.59n \rceil$ bits. This encoding gives us an upper bound on the number of distance-hereditary graphs of n vertices: there are at most $2^{\lceil 3.59n \rceil}$ non-isomorphic distance-hereditary graphs with n vertices. Applying the technique to cographs, each cograph of n vertices can be represented in at most $3n$ bits, and hence the number of cographs of n vertices is at most 2^{3n}. These upper bounds are improvements on the previously known upper bounds of $2^{O(n \log n)}$[30,p.20, p.98].

2 Preliminaries

The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N(v) = \{u \in V \mid \{u, v\} \in E\}$. We denote the closed neighborhood $N(v) \cup \{v\}$ by $N[v]$. For a vertex subset U of V, we denote the set $\{v \in V \mid v \in N(u)$ for some $u \in U\}$ by $N(U)$, and the set $\{v \in V \mid v \in N[u]$ for some $u \in U\}$ by $N[U]$. A vertex set C is a *clique* iff all pairs of vertices in C are joined by an edge in G. If a graph $G = (V, E)$ itself is a clique, it is said to be *complete* and is denoted by K_n, where $n = |V|$. Given

a graph $G = (V, E)$ and a subset U of V, the *induced subgraph* by U, denoted by $G[U]$, is the graph (U, E'), where $E' = \{\{u, v\} \in E \mid u, v \in U\}$. For a vertex w, we sometimes denote the graph $G[V \setminus \{w\}]$ by $G - w$ for short. Two vertices u and v are said to be *twins* iff $N(u)\backslash\{v\} = N(v)\backslash\{u\}$. For twins u and v, we say that u is a *strong sibling* of v iff $\{u, v\} \in E$, and a *weak sibling* iff $\{u, v\} \notin E$. We also say *strong* (*weak*) *twins* iff they are strong (weak) siblings. If a vertex v is a strong or weak twin of another vertex u, we simply say that they are twins and v has a sibling u. Since twins are transitive, we say that a vertex set S with $|S| > 2$ comprises strong (weak) twins iff each pair in S consists of strong (weak) twins. In this paper, we will order the vertices. For an ordered vertex set, let S be a set of twins. Then, we call $v \in S$ a *larger* sibling of S if $v > v'$ for some other $v' \in S$ with respect to the ordering. That is, every sibling in S is a larger sibling, except the smallest one, which will be called the *smallest* sibling of S. Note that it is easy to find a larger sibling for given S; take arbitrary two twins, and pick the larger one.

For two vertices u and v, the *distance* between the vertices, denoted by $d(u, v)$, is the minimum length of the paths joining u and v. We extend the neighborhood recursively as follows: for a vertex v in G, we define $N_0(v) := \{v\}$ and $N_1(v) := N(v)$. For $k > 1$, $N_k(v) := N(N_{k-1}(v)) \setminus (\cup_{i=0}^{k-1} N_i(v))$. That is, $N_k(v)$ is the set of vertices of distance k from v.

In this paper, the following graph operations play an important role; (α) pick a vertex x in G and add a new vertex x' with an edge $\{x, x'\}$, (β) pick a vertex x in G and add x' with edges $\{x, x'\}$ and $\{x', y\}$ for all $y \in N(x)$, and (γ) pick a vertex x in G and add x' with edges $\{x', y\}$ for all $y \in N(x)$. For operation (α), we say that the new graph is obtained by attaching a *pendant vertex* x' to the *neck vertex* x. In (β) and (γ), it is easy to see that x and x' are strong and weak twins, respectively. In this case, we say that the new graph is obtained by *splitting* the vertex x into strong and weak twins. It is known that the class of *cographs* is characterized by the above operations as follows (see [31] for characterizations of cographs):

Theorem 1. *A connected graph G with at least two vertices is a cograph iff G can be obtained from K_2 by a sequence of operations (β) and (γ).*

These operations are also used by Bandelt and Mulder to characterize the class of *distance-hereditary graphs*[18].

Theorem 2. *A connected graph G with at least two vertices is distance-hereditary iff G can be obtained from K_2 by a sequence of operations (α), (β), and (γ).*

We add one pendant vertex in (α), and split a vertex into two siblings in (β) and (γ). In this paper, we also use the following generalized operations for $k \geqslant 1$; (α') pick a neck x in G and add k pendants to x; (β') pick a vertex x in G and split it into $k+1$ strong siblings; and (γ') pick a vertex x in G and split it into $k+1$ weak siblings.

For a vertex set $S \subseteq V$ of $G = (V, E)$, the *contraction* of S into $s \in S$ is obtained as follows: 1) for each edge $\{v, u\}$ with $v \in S \setminus \{s\}$ and $u \in V \setminus S$, add an edge $\{u, s\}$ to E; 2) replace multiple edges by a single edge; and 3) remove all vertices in $S \setminus \{s\}$ from V and their associated edges from E.

Let $v_1, v_2, \ldots, v_n$ be an ordering of a vertex set V of a connected distance-hereditary graph $G = (V, E)$. Let G_i denote the graph $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ for each i. The ordering is a *pruning sequence* iff G_{n-1} is K_2 and G_i can be obtained from G_{i+1} by either attaching a pendant vertex v_i or splitting some vertex $v \in G_{i+1}$ into twins v and v_i for each $i < n - 1$. (In other words, G_{i+1} is obtained from G_i by either pruning a pendant vertex v_i or contracting some twins v_i and $v \in G_{i+1}$ into v for each $i < n - 1$.) For a connected cograph G, the pruning sequence of G is defined similarly only by splitting vertices.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* iff there is a one-to-one mapping $\phi : V \to V'$ such that $\{u, v\} \in E$ iff $\{\phi(u), \phi(v)\} \in E'$ for every pair of vertices $u, v \in V$. We say that $G \sim G'$ if they are isomorphic.

2.1 Open and Closed Prefix Trees and Basic Operations

Here, we introduce the prefix trees of open and closed neighbors. These are used for detecting strong twins, weak twins, pendants, and necks efficiently. The details of the notion of a prefix tree, which is also called a trie, can be found in standard textbooks (e.g., [14]).

Let V be an ordered set of vertices (hereafter, we assume that the vertices are numbered from 1 to $|V|$ in some way). A *prefix tree* for a family of subsets of V is a rooted tree T satisfying the following conditions (the vertices of prefix trees are called *nodes* to distinguish them from the vertices in V). Except the root, each node in T is labeled by a vertex in V, and some nodes are (doubly) pointed to by vertices in V. The labels of a prefix tree satisfy the following two conditions; 1) if a node with label v is the parent of a node of label v', then $v' > v$, and 2) no two children of a node have the same label. Note that two or more nodes in T can have the same label (the name of a vertex of V), but each vertex in V has exactly one pointer to a node in T.

We will maintain $N(v)$ and $N[v] = N(v) \cup \{v\}$ for all

vertices in G by using two prefix trees as follows. The path of a prefix tree from a node x to the root gives a set of labels (or vertex set in G), denoted by $set(x)$. If x is pointed to by a vertex v, we consider that $set(x)$ is associated with v. In this manner, we can use the two prefix trees to represent two families of open neighborhoods and closed neighborhoods by considering the neighbor set as the associated set. We call these prefix trees the *open* and *closed prefix trees*, and denote them by $T(G)$ and $T[G]$, respectively. Intuitively, sorting $N(v)$ and $N[v]$ defines a unique sequence $L(v)$ of vertices v for each $v \in V(G)$, and merging common prefixes of those sequences $\{L(v) \mid v \in V(G)\}$ generates the open prefix tree $T(G)$ and the closed prefix tree $T[G]$, respectively.

The prefix trees $T(G)$ and $T[G]$ for the distance-hereditary graph G in Fig.1(a) are depicted in Fig.2. Each square is a node labeled by a vertex in G except the root. Each circle indicates a vertex in G, and the thick arrows are (double) pointers to the corresponding node. That is, a vertex in a circle has a neighbor set that appears on the path from the pointed node to the root, and they are incrementally ordered from the root to the node.

We can make it so that every leaf of the prefix tree is pointed to by at least one vertex (since leaves pointed to by no vertices are redundant). Thus, the size of the prefix tree is $O(n+m)$. Here, we clarify the data structure for the prefix tree. A prefix tree is represented by the set of its nodes, and each node has the pointer to its parent. The children of a node x is maintained by a doubly linked list of pointers to the children, and the list is in x. Thus, we can get the parent of a node in constant time, but finding a specified child (by a label) takes linear time in the number of children. We assume that the pointers between a parent and its children are doubly linked. Hence, when an algorithm deals with a child v, the algorithm can remove the pointer to v in $O(1)$ time in the doubly linked list to the children located in its parent.

Lemma 3. *For any given graph $G = (V, E)$, $T[G]$ and $T(G)$ are constructed in $O(n+m)$ time and space.*

Proof. In general, a prefix tree of a given family can

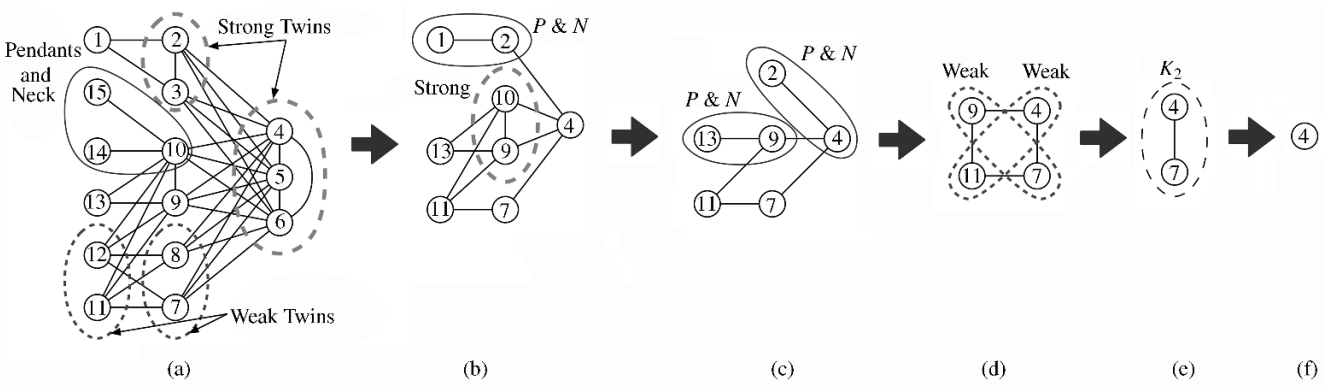

Fig.1. Distance-hereditary graph and its contracting/pruning process.

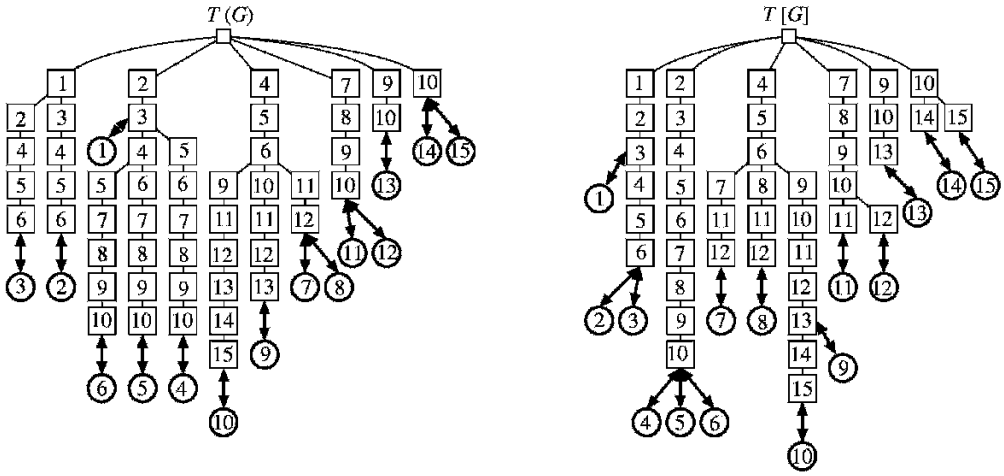

Fig.2. Two prefix trees for G in Fig.1(a).

be constructed in linear time. To make the paper self-contained, we describe the construction algorithm for the open prefix tree in Algorithm 1. Note that while every vertex in G points to exactly one node in T, some nodes x in T have a list of vertices in G pointing to x.

Algorithm 1. Construction of the Open Prefix Tree
Input: graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$;
Output: $T(G)$;

```
   // In Steps 1∼4, sort neighbors of all vertices
      by bucket sort
1  P(v) := ∅ for each v ∈ V;
2  foreach v = 1, 2, ..., n do
3  |  insert v to P(u) for each u ∈ N(v);
4  end
   // Hereafter, construct prefix tree level-by-
      level
5  initialize T as a tree with one root node (initially)
   pointed to by every vertex of G;
6  foreach k = 0, 1, 2, ..., max_{v∈V} |P(v)| do
7  |  foreach node x on the k-th depth of T (in
   |  increasing order) do
8  |  |  foreach vertex v pointing to x (in increasing
   |  |  order) do
9  |  |  |  if |P(v)| > k then let u be the (k+1)-th
   |  |  |  vertex in P(v);
10 |  |  |  if x has no child with label u yet then
10 |  |  |  create the child y of x with label u;
11 |  |  |  update so that v points to y;
12 |  |  end
13 |  end
14 end
15 return T as T(G).
```

The computation time of Algorithm 1 is $O(n + m)$. The first loop sorts the neighborhood of each vertex by bucket sort. The second loop constructs the prefix tree level-by-level. In the k-th loop, we create all the children of the nodes on the $(k-1)$-th depth. This is done by making a child each k-th neighbor in $P(v)$ for each $v \in V$.

Note that Step 10 can be done in $O(1)$ time; we put a "mark" y on vertex u in V when node y is made. All marks of vertices in V for x are cleared after Step 12.

The closed prefix tree can be constructed in a similar way. □

From the definitions, we can immediately have the following observations.

Observation 4. *Let u and v be any vertices in G. Then,* 1) *u is the pendant of the neck v iff a child of the root of $T(G)$ has label v and is pointed to by u;* 2) *u and v are strong twins iff u and v point to the same node in $T[G]$; and* 3) *u and v are weak twins iff u and v point to the same node in $T(G)$.*

We have to update the prefix trees efficiently. Hence, Algorithm 2 for removing a node from a prefix tree is a key procedure.

Algorithm 2. Delete a Vertex from a Prefix Tree
Input: open (or closed) prefix tree $T(G)$ (or $T[G]$), vertex w in V to be deleted;
Output: open (or closed) prefix tree $T(G - w)$ (or $T[G - w]$);

```
1  let x be the unique node pointed to by w;
2  While x is a leaf node and pointed to by no vertex
   except w do delete x and set x to be its own parent;
3  foreach node x with label w do
4  |  let y be the parent of node x;
5  |  foreach vertex v pointing to node x
   |  do update so that v points y;
6  |  remove x from the list of children in y;
7  |  let L_x and L_y be the lists of children nodes of x
   |  and y, respectively;
8  |  merge L_x and L_y to obtain the list of children
   |  of y;
9  |  delete x from T;
10 end
11 return T.
```

For a graph $G = (V, E)$ and a vertex w, Algorithm 2 correctly obtains $T(G - w)$ and $T[G - w]$ from $T(G)$ and $T[G]$, respectively. We now analyze the complexity of Algorithm 2. The node x pointed to by the vertex w has an ancestor with label v iff $w \in N(v)$ or $w \in N[v]$. Thus, the number of ancestors of the node pointed to by w is at most $|N[w]|$, and hence Steps 1 and 2 can be done in $O(|N[w]|)$ time. Moreover, the number of nodes with label w is bounded by $|N[w]|$. Hence, the loop from Steps 3∼10 will be repeated at most $|N[w]|$ times. Steps 4, 6, 7, 9, and 10 can be done in $O(1)$ time, and Step 5 can be done in $O(|N[w]|)$ time. Therefore, our main task is to evaluate Step 8, which merges two lists of nodes x and y. Generally, lists L_x and L_y may contain nodes having the same label. In such cases, we have to unify the nodes by merging the two lists L_x and L_y. We here call this operation *unification*. However, we will show in Lemmas 22 and 24 that in our algorithm L_x and L_y never have a common node; thus, no

unification occurs. This is the key to our linear time algorithm.

3 Canonical Trees

In this section, we introduce the notion of the DH-tree, which is a canonical tree representation of a distance-hereditary graph. First, we define the DH-tree derived from a distance-hereditary graph G. Although it is canonical, it is still redundant. Hence, we next introduce the normalized DH-tree, which will be used as the canonical and compact representation of a distance-hereditary graph.

Any cograph is a distance-hereditary graph. The normalized DH-tree for a cograph G coincides with the cotree, which is a known tree representation of a cograph. That is, the normalized DH-tree is a natural generalization of the cotree.

3.1 DH-Tree Derived from a Distance-Hereditary Graph

We will deal with K_1 (single vertex) and K_2 (two vertices joined by an edge) as special distance-hereditary graphs. Hence, hereafter, we assume that $G = (V, E)$ is a connected distance-hereditary graph that contains at least three vertices. For given distance-hereditary graph $G = (V, E)$, we define three families of vertex sets as follows:

$$\begin{aligned}\mathcal{S} :=& \{S \mid x, y \in S \text{ if } N[x] = N[y] \text{ and } |S| \geqslant 2\},\\ \mathcal{W} :=& \{W \mid x, y \in W \text{ if } N(x) = N(y), |W| \geqslant 2,\\ & \text{and } |N(x)| = |N(y)| > 1\},\\ \mathcal{P} :=& \{P \mid x, y \in P \text{ if } x \text{ is a pendant vertex}\\ & \text{and } y \text{ is its neck}\}.\end{aligned}$$

Note that when y is the neck of two pendant vertices x_1 and x_2, $\mathcal{P}$ contains the set P with $\{x_1, x_2, y\} \subseteq P$. Moreover, we have the following observation.

Lemma 5. *For each P in $\mathcal{P}$, P contains exactly one neck with associated pendants.*

Proof. Since each pendant has degree 1, if P contains two necks y and y', y is a pendant of the neck y', and vice versa. This implies that G is K_2, which contradicts to the assumption. □

Now we show that the families give disjoint sets. More precisely, for any pair of sets S_1 and S_2 in $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$, $S_1 \cap S_2 = \varnothing$. (Note that we have $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \subseteq V$, or some vertices may not belong to any vertex set.)

Lemma 6. *Let v be any vertex in a distance-hereditary graph G. Then v belongs to either* 1) *exactly one set in $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$, or* 2) *no set in the families.*

Proof. If v belongs to no set, we have nothing to do. Hence, we assume that v belongs to at least one set. We now have three cases.

1) v is in a set P in $\mathcal{P}$. Then we have two subcases. First, we assume that v is a pendant. Then there is a unique neck u in P, and by Lemma 5, it is easy to see that $v \notin P'$ in $\mathcal{P}$ with $P \neq P'$. Here, $|N(v)| = |\{u\}| = 1$; hence $v \notin W$ for any $W \in \mathcal{W}$. On the other hand, $N[v] = \{u, v\}$; hence there is no other vertex v' with $N[v'] = N[v]$ except $v' = u$. However, $N[u] = N[v]$ implies that $G \sim K_2$, which is a contradiction. Thus $v \notin S$ for any $S \in \mathcal{S}$.

Next, we assume that v is a neck in some P in $\mathcal{P}$. Then there is a pendant u of v. According to a similar argument to the one above, there is no other set $P' \in \mathcal{P}$ that contains v. Moreover, the only neighbor of u is v. Hence there is no other vertex v' with $N(v) = N(v')$ and $N[v] = N[v']$ (except $v' = u$, which implies $G \sim K_2$, a contradiction). Thus, P is the only set containing v.

2) v is in S for some set S in $\mathcal{S}$. By 1), there is no set $P \in \mathcal{P}$ that contains v. We first assume that there is another set $S' \in \mathcal{S}$ with $v \in S'$. Then it is easy to see that for any $u \in S$ and $u' \in S'$, $N[u] = N[u'](= N[v])$. Hence we have $S = S'$, which is a contradiction. Now we assume that there is another set $W \in \mathcal{W}$ with $v \in W$. By assumption, $N[v] = N[u]$ for some u in S, and $N(v) = N(w)$ for some w in W. Since $u, v \in S$, $\{u, v\} \in E$. Therefore, we have $N(w) = N(v) \cup \{u\}$ and hence $\{u, w\} \in E$. This contradicts the fact that $N[v] = N[u]$ and $\{v, w\} \notin E$.

3) v is in W for some set W in $\mathcal{W}$. By 1) and 2), there is no set $P \in \mathcal{P}$ and $S \in \mathcal{S}$ that contains v. To derive a contradiction, we assume that another W' in $\mathcal{W}$ contains v. Then it is easy to see that $W = W'$, which is a contradiction.

Therefore, the family $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$ define disjoint sets of V. □

Lemma 7. *For any distance-hereditary graph G, $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \varnothing$.*

Proof. By Theorem 2, G can be generated by a sequence of operations. For the last operation of the sequence, we have a non-empty set. □

Here, we introduce the notion of the DH-*tree derived from a distance-hereditary graph G*, which is a rooted ordered tree where each inner node has a label (we again use the notation "node" for a DH-tree (and prefix tree) to distinguish from a "vertex" in G). The label of an inner node is one of $\{\mathtt{s}, \mathtt{w}, \mathtt{p}\}$, which indicate strong twin, weak twin, and pendant with neck, respectively.

Given a distance-hereditary graph $G = (V, E)$, the DH-tree derived from G is defined recursively from

leaves to the root. We have three basic cases:

1) $G = K_1$: the DH-tree derived from K_1 is defined by a single root with no label.

2) $G = K_n$: when $G \sim K_n$ with $n \geqslant 2$, the DH-tree of G is defined by a single root with label `s` and n leaves with no labels. The tree represents that K_n can be obtained from a single vertex K_1 by splitting it into n strong siblings.

3) $G = S_n$, where S_n is a *star* with $n > 2$ vertices that consist of a center vertex with $n-1$ pendant vertices. In this case, we define the DH-tree derived from G as a tree with a single root with label `p` and n leaves with no labels. Note that the tree is ordered. The leftmost child of a node with label `p` indicates the neck. That is, the leftmost leaf corresponds to the center of the star, and the $n-1$ leaves correspond to the $n-1$ pendants.

Note that the cases are disjoint. In particular, K_2 is a clique, and not a star. Hence, the root with label `p` of a DH-tree has at least three children. Note also that the number of leaves of the tree is the number of vertices in G. This fact is an invariant for the DH-tree.

We now define the DH-tree $\mathcal{T}$ derived from $G = (V, E)$ with $|V| = n > 2$ in the general case. We assume that G is neither K_n nor a star of n vertices. We start with n leaves of $\mathcal{T}$; they are initially independent. Since each leaf in $\mathcal{T}$ corresponds to a vertex v in G, we identify the leaf with v hereafter. Then, by Lemmas 6 and 7, we can group the leaves into three kinds of families $\mathcal{S}$, $\mathcal{W}$, and $\mathcal{P}$ with $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \varnothing$. Let S be any set in $\mathcal{S}$. We make a common parent with label `s` of the leaves. We repeat this process for each S in $\mathcal{S}$. For each set W in $\mathcal{W}$, we similarly make a common parent with label `w`. Let P be any set in $\mathcal{P}$. Then, by Lemma 5, P contains exactly one neck v and some pendants u. We make a common parent with label `p` for them, and make the neck v the leftmost child of the parent. The pendants are placed to the right of the neck in arbitrary ordering.

Next, we contract each vertex set U in $\mathcal{S} \cup \mathcal{W}$ into any vertex $u \in U$ on G. Each vertex u corresponds to a parent in the resultant $\mathcal{T}$ and we identify these correspondences. For each $P \in \mathcal{P}$, we also prune all pendant vertices except the unique neck in P. The neck corresponds to the parent of the nodes in P.

We repeat the process until the resultant graph becomes one of the basic cases, and we obtain the DH-tree $\mathcal{T}$ derived from $G = (V, E)$.

An example of a distance-hereditary graph G and the DH-tree derived from G are depicted in Figs. 1 and 3, respectively. The twins are contracted by removing all siblings except the smallest one. Each node in the DH-tree corresponds to a vertex in the distance-hereditary graph and is depicted in Fig.3. Note that the DH-tree itself does not store information.

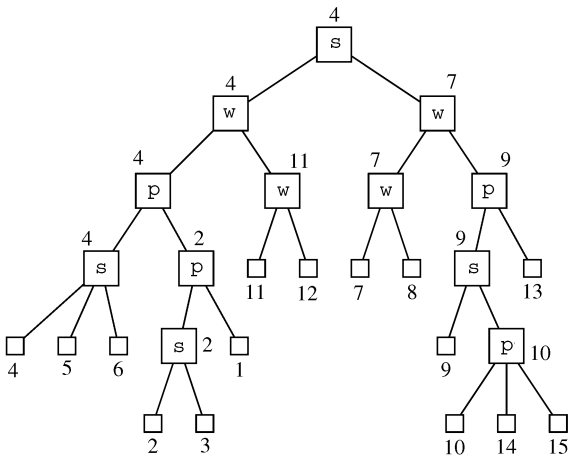

Fig.3. DH-tree $\mathcal{T}$ derived from the graph in Fig.1(a).

Theorem 8. *We can construct the* DH*-tree* $\mathcal{T}$ *derived from* G *iff* G *is a distance-hereditary graph.*

Proof. Due to Bandelt and Mulder in [18], G is distance-hereditary iff G has a pruning sequence. Hence, we can prove the theorem by using simple induction for the number of the vertices of G with Lemmas 6 and 7. □

Corollary 9. 1) *For any distance-hereditary graph* G*, the* DH*-tree derived from* G *is uniquely determined.*

2) *The* DH*-tree* $\mathcal{T}$ *derived from a distance-hereditary graph* $G = (V, E)$ *requires* $O(|V|)$ *space.*

Proof. 1) The corollary is true from the fact that the families $\mathcal{S}$, $\mathcal{W}$ and $\mathcal{P}$ are uniquely determined.

2) By definition, the number of leaves of $\mathcal{T}$ is equal to $|V|$. Moreover, every inner node of $\mathcal{T}$ has at least two children. Thus, the number of inner nodes is at most $|V|$, which implies the corollary. □

The following characterization will be used later.

Lemma 10. *Let* G *be a distance-hereditary graph that contains at least two vertices, and* $\mathcal{T}$ *be the* DH*-tree derived from* G*. Let* d *be the diameter of* $\mathcal{T}$*. Then we have the following*:

(a) $d \geqslant 2$,
(b) *each inner node has at least two children,*
(c) *each inner node has a label, either* `p`*,* `s` *or* `w`*, and each leaf has no label,*
(d) *the label of the root is* `p` *or* `s`*, and*
(e) *any non-leftmost child of a node with label* `p` *is not* `w`*.*

Proof. The above statements except the last one immediately come from the definition. Statement (e) is true since a weak neighbor of a pendant is always a pendant. □

3.2 Normalized DH-Tree of a Distance-Hereditary Graph

Subsection 3.1 introduced the notion of the DH-tree derived from a distance-hereditary graph $G = (V, E)$. However, such a graph can be redundant.

As a rule (β) can be replaced by (β'), if a node x with the label "w" is the parent of the other nodes y with the label "w," x and y can be weak siblings in the same level (in Fig.4, case (a) can be replaced by (b)). The same reduction can be applied to the nodes with the label "s."

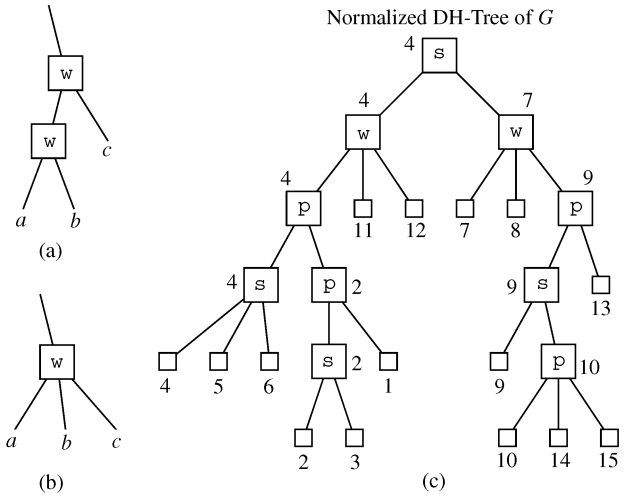

Fig.4. Reduction rule and the compact DH-tree.

Hence, we introduce the notion of the *normalized* DH-tree of a distance-hereditary graph G, which is obtained from the DH-tree derived from G by applying the reduction repeatedly as possible as we can (see Fig.4(c) which is the normalized DH-tree obtained from the DH-tree in Fig.3). Lemma 10 also holds for the normalized DH-tree. Below, we prove two lemmas for a normalized DH-tree.

Lemma 11. *From a given* DH*-tree derived from a distance-hereditary graph $G = (V, E)$, the normalized* DH*-tree of G can be constructed in $O(|V|)$ time and space.*

Proof. The reduction can be done by using the standard depth first search. Hence, the normalized DH-tree $\mathcal{T}$ of G can be obtained from the DH-tree $\mathcal{T}'$ derived from G in $O(|\mathcal{T}|) = O(|\mathcal{T}'|) = O(n)$ time and space. □

The *height* of a node on a rooted tree is the distance from it to the farthest node in its descendants. (Hence the height of a leaf is 0.)

Lemma 12. *For any distance-hereditary graph $G = (V, E)$, let $\mathcal{T}$ be the* DH*-tree derived from G. Let x be any node in $\mathcal{T}$ and y be its parent. Assume that x and y have the same label and are reduced in the normalized tree. Then, the height of y does not change in the normalized* DH*-tree of G.*

Proof. Without loss of generality, we assume that x and y have the same label w. If all children of y have the same label w, they form a set of weak twins, which is a contradiction. Hence y has at least one child z that has a different label. Among them, we assume that z has the largest height. We show that the height of y is given by z, not by x. To derive a contradiction, we assume that z is lower than x. Recall that the DH-tree is derived from leaves to root. Hence, when z is lower than x, z and other lower nodes are merged by the algorithm and we obtain y with label w. Then y cannot be the parent of x, which contradicts the assumption that y is the parent of x. Hence, the height of y does not change in the normalized tree. □

Lemma 12 immediately leads to the following lemma.

Lemma 13. *For any distance-hereditary graph G, let $\mathcal{T}$ and $\mathcal{T}'$ be the* DH*-tree derived from G and the normalized* DH*-tree of G, respectively. Then, the roots of $\mathcal{T}$ and $\mathcal{T}'$ have the same height.*

Proof. To prove this lemma, all we need to do is use Lemma 12 and simple induction on the height of the roots of trees. □

In order to characterize the normalized DH-tree, we begin with some definitions. Let x be a node of a normalized DH-tree having label p. The leftmost child of p is called a *neck child*, and the other children are called *pendant children*. For any node x in a (normalized) DH-tree $\mathcal{T}$, let $C(x)$ denote the set of children of x and $L(x)$ denote the leaves in $C(x)$. Then, a *peel operation* at a node x in $\mathcal{T}$ is defined as follows; 1) if $L(x) = C(x)$, remove all children of x; 2) if $L(x) \subset C(x)$ and $L(x) \in \mathcal{W} \cup \mathcal{S}$, remove all children of x except one leaf; and 3) if $L(x) \subset C(x)$ and $L(x) \in \mathcal{P}$, remove all children of x. The peel operation for $\mathcal{T}$ is defined by the union of peel operations at all inner nodes in $\mathcal{T}$. Intuitively, a peel operation corresponds to a contraction of twins in a set U of $\mathcal{W} \cup \mathcal{S}$ or pruning pendants in a set P of $\mathcal{P}$. Hence, we have the following observation.

Observation 14. *Let $G = (V, E)$ be a distance-hereditary graph, $\mathcal{T}$ be the normalized* DH*-tree of G, and $\mathcal{S}$, $\mathcal{W}$, and $\mathcal{P}$ be the families of vertex sets defined by twins and pendants on G. Let G' be the distance-hereditary graph which is obtained from G by contracting all twin sets in $\mathcal{S} \cup \mathcal{W}$ and pruning pendants in $\mathcal{P}$. Then, the normalized* DH*-tree of G' coincides with the* DH*-tree obtained from $\mathcal{T}$ by performing the peel operation on $\mathcal{T}$.*

Lemma 15. *Let G be a distance-hereditary graph*

that contains at least two vertices, and $\mathcal{T}$ be a normalized DH*-tree. Let d be the diameter of $\mathcal{T}$. Then the following statements are true*:

(a) *a parent and its children do not have the same label if the parent has label* `s` *or* `w`,

(b) *the root is a center of the tree,*

(c) *if the root has two children, the label of the root is* `s`*, the labels of the children are both* `w` *or both* `p`*, and d is even, and*

(d) *if the label of the root is* `p`*, at least two non-leftmost children of the root have the maximum height among the children of the root.*

Proof. Statement (a) is immediately by the definition of a normalized DH-tree. With careful analysis of the construction of the DH-tree derived from G, we can see that any inner node x has at least two children of the same height that gives the height of x. Hence the root of the DH-tree derived from G is a center of the tree. Thus, we can use Lemmas 12 and 13 to prove (b).

To prove (c), we suppose that the root has two children. Then by the same argument for (a), the two children have the same height that gives the height of the root. Hence, d is even. By definition, the root has label `s`, and the labels of its children are `p` or `w`. If the root has label `s` and two children have different labels `p` and `w`, this implies that in the last step of the construction of DH-tree, the last graph would be a star, which is a contradiction.

The last statement (d) can be obtained from a similar argument to the one for (a); the height of the root is given by at least two non-leftmost children. □

Now we can state the compact and canonical representation of a distance-hereditary graph.

Theorem 16. *The normalized* DH*-tree of a connected distance-hereditary graph is canonical. That is, the normalized* DH*-tree $\mathcal{T}$ for any given connected distance-hereditary graph G is uniquely determined, and the original distance-hereditary graph G is uniquely constructed from the normalized* DH*-tree $\mathcal{T}$ up to isomorphism.*

Proof. By Corollary 9-1), the DH-tree $\mathcal{T}'$ derived from a given distance-hereditary graph G is uniquely determined. Moreover, the normalized DH-tree $\mathcal{T}$ does not change with the order of the normalization process. Hence, the normalized DH-tree $\mathcal{T}$ for G is uniquely determined from G.

Let $\mathcal{T}$ be the normalized DH-tree which is constructed from a distance-hereditary graph G. We show that the original G can be reconstructed from $\mathcal{T}$ by induction on the diameter d of $\mathcal{T}$. Note that when $\mathcal{T}$ contains just the root, we have $d = 0$. Hereafter, we assume $d \geqslant 2$ from Lemma 10(a). Next we assume that $d = 2$. If G is K_n or a star with $n > 1$ vertices, it is easy to see that G can be uniquely reconstructed from $\mathcal{T}$. Suppose that G is neither K_n nor a star with n vertices. Then, $\mathcal{T}$ has diameter 2 if and only if V produces exactly one set in $\mathcal{S}$, $\mathcal{W}$, or $\mathcal{P}$. However, $V \in \mathcal{S}$ implies G is K_n, $V \in \mathcal{W}$ implies G is an independent set of n vertices, and $V \in \mathcal{P}$ implies G is a star with n vertices, which are contradictions. Thus, G is one of the basic cases when $d \leqslant 2$.

Let $\mathcal{T}$ have diameter $d > 2$, and assume that any DH-tree $\mathcal{T}'$ of diameter less than d can uniquely generate the corresponding distance-hereditary graph G'. Let L' be the set of leaves in $\mathcal{T}$. Among them, let L be the set of leaves that give the diameter d of G; that is, for each $v \in L$, there is another vertex u (in L) with $d(u, v) = d$. The set L can be partitioned into distinct sets $L_1, L_2, \ldots, L_k$ such that L_i contains the leaves with a common parent. Then, each common parent of a set L_i is labeled by `w`, `s`, or `p`. Recall that, at first, each leaf corresponds to a vertex in G.

Let $\mathcal{T}''$ be the DH-tree obtained by peeling $\mathcal{T}$. Then, from Observation 14, we have two cases: the diameter $d'' < d$ and $d'' = d$. In either case, as similar discussion to the one in the proof of Lemma 15 shows that G is uniquely reconstructed from $\mathcal{T}$. □

Corollary 17. *The normalized* DH*-tree $\mathcal{T}$ for a distance-hereditary graph $G = (V, E)$ requires $O(|V|)$ space.*

3.3 Cotree for a Cograph

By the characterizations in Theorems 1 and 2, the normalized DH-tree for a cograph only consists of the nodes of labels `s` and `w`. This notion of the tree representation for a cograph is known as the cotree, and appears in many studies, e.g., [15, 16, 31]. We note that only the nodes of label `p` require the ordering of children in the normalized DH-tree. Hence the cotree of a cograph is a labeled rooted non-ordered tree.

4 Levelwise Laminar Ordering

For maintenance of a pendant vertex in a distance-hereditary graph, we introduce a technical vertex ordering, called *levelwise laminar ordering*. Note that for any distance-hereditary graph G and r in G, $N_k(r)$ induces a (not necessarily connected) cograph, and this fact plays an important role in previous papers[15,18]. The levelwise laminar ordering is weaker than the fact.

We first define the notations $N_v^+(u)$, $N_v^-(u)$, and $N_v^0(u)$ as follows. We fix any vertex v in $G = (V, E)$. Then, for each $u \in V$, $N_v^+(u)$ denotes the set $N_{d(u,v)+1}(v) \cap N(u)$, $N_v^-(u)$ denotes the set

$N_{d(u,v)-1}(v) \cap N(u)$, and $N_v^0(u)$ denotes the set $N_{d(u,v)}(v) \cap N(u)$. We define $N_v^-(v) := \varnothing$ for any $v \in V$.

Let V be a set of n vertices. Two sets X and Y are said to be *overlapping* iff $X \cap Y \neq \varnothing$, $X \setminus Y \neq \varnothing$, and $Y \setminus X \neq \varnothing$. A family $\mathcal{F} \subseteq 2^V$ is said to be *laminar* iff $\mathcal{F}$ contains no overlapping sets; that is, any pair of two distinct sets X and Y in $\mathcal{F}$ satisfies either $X \cap Y = \varnothing$, $X \subseteq Y$, or $Y \subseteq X$. Then any distance-hereditary graph has the following laminar structure.

Lemma 18[18, Theorem 3(4)]. *Let $G = (V, E)$ be a distance-hereditary graph. Let r be any vertex in V. Then the family $\{N_r^-(v) \mid v \in N_k(r)\}$ is laminar for any $k \geqslant 1$.*

Lemma 18 is proven in [18, Theorem 3(4)]; this lemma is one of the five properties that characterize the class of distance-hereditary graphs. In other words, Lemma 18 is necessary, but not sufficient condition for distance-hereditary graphs. For example, C_6 also has the laminar property although it is not distance-hereditary.

Let $G = (V, E)$ be a distance-hereditary graph, and r be any vertex in V. Hereafter, we fix r and call it the *root* of the ordering. Now, we introduce a new notion of an ordering $(v_1 = r, v_2, \ldots, v_n)$ of vertices in V. We write $v_i < v_j$ for two vertices v_i and v_j if $i < j$. If a vertex ordering satisfies the following conditions, we call it a *levelwise laminar ordering*:

(L1) For any $v_i \in N_k(r)$ and $v_j \in N_{k'}(r)$, $i < j$ holds if $0 \leqslant k < k'$.

(L2) For any $v_i, v_j \in N_k(r), k \geqslant 1$, $i < j$ holds if $N_r^-(v_j) \subset N_r^-(v_i)$.

(L3) Let v_i, v_j be any vertices in $N_k(r)$, $k \geqslant 1$, $i < j$ such that $N_r^-(v_i) = N_r^-(v_j)$. Then we have $N_r^-(v_i) = N_r^-(v_j) = N_r^-(v)$ for all vertices $v_i < v < v_j$.

By Lemma 18, any distance hereditary graph has a levelwise laminar ordering.

Lemma 19. *Let u be a pendant of a neck v. When u is not the root of the levelwise ordering, $v < u$.*

Proof. The only neighbor of u is v. Hence u will not be numbered before v when u is not the root. □

Lemma 20. *The levelwise laminar ordering of a distance-hereditary graph $G = (V, E)$ can be computed in $O(|V| + |E|)$ time and space.*

Proof. First, we fix the root r. Then we order the vertices by the standard breadth first search from r by using a queue as follows: first, the queue Q is initialized by r, and while Q is not empty, we pick up the first element v from Q and number it, and put the unnumbered vertices in $N(v) \setminus Q$ into the last part of Q. This proves (L1). Next, for (L2), we compute the ordering of the vertices in $N_k(r)$ for each $k = 1, 2, \ldots$. For each k, we have to solve the following subproblem for $X = N_{k-1}(r)$ and $Y = N_k(r)$:

Input: Bipartite graph $G' = (X \cup Y, E')$ with $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_m\}$, and $E' \subseteq X \times Y$.
Output: If there are y_i, y_j in Y such that $N(y_i)$ and $N(y_j)$ overlap, output "No," else output the ordering over Y such that $y_i < y_j$ if $N(y_j) \subset N(y_i)$.

If the problem can be solved in $O(|X| + |Y| + |E'|)$ time and space, we have (L2) in linear time. The vertices in X and Y are maintained in the usual arrays. To simplify the algorithm, we add a universal vertex y_0 to Y such that $N(y_0) = X$. We first sort $Y' := \{y_0, y_1, \ldots, y_m\}$ such that $|N(y_0)| \geqslant |N(y_1)| \geqslant |N(y_2)| \geqslant \cdots \geqslant |N(y_n)|$ by bucket sort. (Ties are broken by the original ordering.) Then, if Y can be sorted in a levelwise laminar ordering, the ordering already satisfies (L2) since $N(y) \subseteq N(y')$ implies $|N(y)| \leqslant |N(y')|$. Hence, it is sufficient to check if they do not overlap. To check that, we define the last(x) for each vertex x in X; last$(x) = y_j$ indicates that the last vertex set containing x is $N(y_j)$. Hence, we can determine if some $N(y_{j'})$ overlaps $N(y_j)$ with $y_j < y_{j'}$, since $N(y_{j'})$ contains two vertices x and x' with last$(x) \neq$ last(x'). The algorithm is as follows:

Algorithm 3. Laminar Ordering in a Level
Input: a bipartite graph $G'' = (X \cup Y', E'')$ with $X = \{x_1, x_2, \ldots, x_n\}$, $Y' = \{y_0, y_1, y_2, \ldots, y_m\}$, and $E'' = E' \cup \{\{y_0, x_i\} \mid 1 \leqslant i \leqslant n\}$.
Output: $Y' \setminus \{y_0\}$.

```
1  sort Y' such that |N(y_0)| ⩾ |N(y_1)| ⩾ |N(y_2)| ⩾ ⋯
   ⩾ |N(y_n)| by bucket sort;
2  foreach i = 1, 2, ..., n do set last(x_i) := y_0;
3  foreach j = 1, 2, ..., m do
4      check if all vertices x in N(y_j) have the same
       last(x), and output "No" if not;
5      update last(x) := y_j for all vertices x in N(y_j);
6  end
7  return Y' \ {y_0}.
```

Showing the correctness of Algorithm 3 is easy. It is not difficult to see that Algorithm 3 runs in $O(|X| + |Y'| + |E''|)$ time and space. Moreover, the algorithm is stable for the breadth first search; ties are broken in the breadth first search manner, which implies (L3).

Hence, we can compute a levelwise laminar ordering of a distance-hereditary graph $G = (V, E)$ in $O(|V| + |E|)$ time and space. □

Note that the levelwise laminar ordering is a partial order, and it is weaker than the LexBFS order.

Levelwise laminar ordering will play an important role for the deletion of pendants. To use the ordering in the deletion algorithm, we have to show that contraction of twins and deletion of pendants do not spoil the ordering.

Lemma 21. *Let $(v_1, v_2, \ldots, v_n)$ be a levelwise laminar ordering of a distance-hereditary graph $G = (V, E)$. Then $G - v_i$ is a distance-hereditary graph and $(v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$ is a levelwise laminar ordering of $G - v_i$ if v_i is either* 1) *a pendant in G, or* 2) *a larger sibling in G.*

Proof. By Theorem 2, $G - v_i$ is a distance-hereditary graph. Hence we show that $(v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$ is a levelwise laminar ordering of $G - v_i$ in each case. We first observe that the deletion of v_i has some effect on the vertices only in $N(v_i)$.

1) When v_i is a pendant with $i > 1$, $N_r^-(v)$ does not change for all vertices v in $V \setminus \{v_i\}$. If $v_i = v_1$, $(v_2, v_3, \ldots, v_n)$ is a levelwise laminar ordering of $G - v_i$ since $N(v_1) = \{v_2\}$.

2) Let w be a smaller sibling of v_i. We have two cases. First, we assume that $w \neq r = v_1$. Since w and v_i are twins, $d(r, w) = d(r, v_i)$. Thus w and v_i are in $N_k(r)$ for some k. Hence, for any vertex x, $N_r^-(x)$ contains both w and v_i or none of them. Therefore, removing v_i has no effect on the inclusion relation between $N_r^-(x)$ and $N_r^-(x')$ for any x and x'. The other case is that $w = r = v_1$. Then $v_i \in N_1(r)$ and $N[w] = N_0(r) \cup N_1(r)$ if $S \in \mathcal{S}$, or $v_i \in N_2(r)$ and $N(v_i) = N_1(r)$ if $S \in \mathcal{W}$. In any case, removing v_i does not violate the properties (L1), (L2), and (L3). □

5 Linear Time Construction of Canonical Trees

In this section, we give linear time algorithms for constructing the cotree of a cograph and the DH-tree of a distance-hereditary graph. From Lemma 6 and Observation 4, a set $W \in \mathcal{W}$ (resp., $S \in \mathcal{S}$) corresponds to a set of vertices pointing to the same node in $T(G)$ (resp., $T[G]$). From Lemma 5, a set $P \in \mathcal{P}$ contains one neck v. Then, by Lemmas 5 and 6, and Observation 4, the set P corresponds to a node x of depth 1 in $T(G)$ such that 1) x has label v, 2) at least one vertex in G points to x, and 3) all vertices in $P \setminus \{v\}$ point to x. The outline of the construction of the canonical tree for a cograph or a distance-hereditary graph is:

(0. sort the vertices in the levelwise laminar ordering,)
1. construct the open and closed prefix trees,
2. if $G \sim K_n$ or G is a star, complete $\mathcal{T}$ and halt,
3. produce nodes of $\mathcal{T}$ for vertices in $\mathcal{P} \cup \mathcal{S} \cup \mathcal{W}$,
4. for each set $S \in \mathcal{S} \cup \mathcal{W}$, contract larger siblings in S to the unique smallest sibling with the update of prefix trees $T(G)$ and $T[G]$,
(5. for each set $P \in \mathcal{P}$, prune all pendants with the update of prefix trees $T(G)$ and $T[G]$,)
6. go to Step 2.

Steps 0 and 5 are required only for a distance-hereditary graph to remove pendant vertices efficiently. Our algorithm contracts all twins in $\mathcal{S} \cup \mathcal{W}$ and prunes all pendants in $\mathcal{P}$, and then returns to Step 2. That is, if contraction of twins produces a pendant, its pruning is postponed to the next execution of the loop. Now we fix the families $\mathcal{P}$, $\mathcal{S}$, and $\mathcal{W}$. Let w be a vertex which will be removed from G since it is one of twins or a pendant.

5.1 Cotree for a Cograph

Here, we consider the case of a cograph. We have to consider the case in which w is a larger sibling. We have another sibling w' with $w' < w$.

Lemma 22. *Let x be any node of a prefix tree with label w, and y the parent of x. Then x has the largest label (vertex) among the children of y.*

Proof. To derive a contradiction, suppose that there is a child x' of y having a label larger than w. Then, from the definition of prefix tree, no descendant of x' has label w. On the other hand, any vertex in G is adjacent to both w and w', or to neither w nor w'. Thus, there is a vertex v in G such that there is a descendant of x pointed to by v, and $N(v)$ contains both w and w'. Hence, there is an ancestor z of x with label w' since $w > w'$. Now let x'' be any descendant of x' (including x') pointed to by v'. Since x' is in the prefix tree with no redundancy, such a vertex v' should exist. Thus we have $w' \in N[v']$ and $w \notin N[v']$, which contradicts that w and w' are twins. □

By Lemma 22, we can see that no unification occurs in this case. Thus we have the following theorem.

Theorem 23. *When w is a larger sibling of a twin, the prefix trees $T(G - w)$ and $T[G - w]$ can be obtained from $T(G)$ and $T[G]$ in $O(|N(w)|)$ time.*

5.2 DH-Tree for a Distance-Hereditary Graph

Next we consider the case of a distance-hereditary graph. We employ two tricks to remove a pendant w. The first trick is using the levelwise laminar ordering obtained in Step 0. The second trick is to deal with two special sets $N[r]$ and $N(r)$ outside the two prefix trees $T[G]$ and $T(G)$; that is, we remove $N[r]$ and $N(r)$ from $T[G]$ and $T(G)$, respectively, and maintain them in another way. More precisely, the algorithm performs as follows.

1) In Step 0, sort the vertices in a levelwise laminar ordering.

2) In Step 1, construct the prefix trees $T[G]$ and $T(G)$ not including $N[r]$ and $N(r)$.

3) In Step 5, maintain all neighbor sets $N[v]$ and $N(v)$ by the prefix trees $T[G]$ and $T(G)$ except the root r. Maintain two special sets $N[r]$ and $N(r)$ separately.

That is, the prefix trees $T[G]$ and $T(G)$ consist of all vertices (including r) as a label, but the root r points to none of their nodes. Two special sets $N[r]$ and $N(r)$ are maintained by standard doubly linked lists; they are initialized in Step 0 in $O(N[r])$ time and space, and each element is pointed to by the neighbor of r.

Now we are going to remove w from G since it is a twin or pendant. When w is a larger sibling, we have already done so in the previous section. We note that we never have $w = r$ since w is a larger sibling and r has the first index. Thus Lemma 22 also holds in this case, and we have nothing to maintain $N[r]$ and $N(r)$.

5.2.1 Pruning a Pendant

We assume that w is a pendant vertex with neck u such that $w \notin N[r]$ (or equivalently, $w \in N_i(r)$ with $i \geqslant 2$). Since $w \notin N[r]$, we have $u \neq r$. We assume that vertices are levelwise laminar ordered $(r = v_1, v_2, \ldots, v_n)$ from the root r in V. Since $w \neq r$, we have $u < w$ by Lemma 19.

Lemma 24. *Let w be a pendant in $N_i(r)$ with $i \geqslant 2$, and u the neck of w. Algorithm 2 can delete w from $T(G)$ and $T[G]$ with no unification.*

Proof. Let x be any node of label w in $T(G)$ or $T[G]$. We first observe that u is the only neighbor of w. Hence there are three paths containing x of label w in $T[G]$ and $T(G)$, which correspond to $N[w]$, $N[u]$, and $N(u)$. We consider three cases.

1) In $T[G]$, x is produced by $N[w]$. Since $N[w] = \{u, w\}$, u is the only neighbor of w, and $u < w$, it is easy to see that x is the leaf of the path of length 2. Thus, no unification occurs.

2) In $T[G]$, x is produced by $N[u]$. Since the only neighbor of w is u, each node under x has exactly one child, and the leaf of the path is pointed to by u.

If w is the maximum vertex in $N[u]$, x is a leaf and can be removed in $O(1)$ time. Thus we assume that $N[u]$ contains $w' > w$. Let y be the unique child of x, and z the parent of x. Without loss of generality, we assume that w' is the label of the unique child y. When w' is also a pendant, u is the only neighbor of w'. Then z has no child that has the same label w' of y. Hence no unification occurs. Thus, we assume that w' is not a pendant. Combining that w is a pendant in $N_i(r)$ and Lemma 18, we have $w' \in N_i(r)$ and $N_r^-(w') = \{u\}$. We now observe that $u \neq v_1$ since $w \notin N[r]$. Thus $N_r^-(u) \neq \varnothing$. Therefore, there is no other vertex (except u) that is adjacent to all vertices in $N_r^-(u) \cup \{w'\}$. This implies that z has no child of label w; hence, no unification occurs.

3) In $T(G)$, x is produced by $N(u)$. Taking care that $u \in N_i(r)$ with $i > 1$, we can see that all cases are the same as in case 2).

Thus, in all cases, Algorithm 2 deletes w from $T(G)$ and $T[G]$ with no unification. □

5.2.2 Removing Pendant in $N[r]$

Now we turn to the operations needed in the case that w is in $N_i(r)$ for $i = 0, 1$. In other words, we consider the case that the pendant w to be removed is in $N(r)$ or $w = r$. We first assume that $i = 1$, or consequently, a pendant w is in $N(r)$.

Lemma 25. *When w is a pendant in $N(r)$, Algorithm 2 deletes w from G in $O(1)$ time.*

Proof. We can remove w from $N(r)$ and $N[r]$ (which are independent from $T(G)$ and $T[G]$) in $O(1)$ time since w has two pointers to the corresponding data in the linked lists $N(r)$ and $N[r]$. Removing w does not change the root of the levelwise laminar ordering. Hence we do not need to update $T(G)$ and $T[G]$. □

Note that Lemma 25 is quite simple since $N[r]$ and $N(r)$ are not in $T[G]$ and $T(G)$. If $N(r)$ is in $T(G)$ and $N[r]$ is in $T[G]$, a considerable case occurs. When the pendant $w \in N(r)$ to be removed is not maximum in $N(r)$, several unifications will occur. In that case, it is very hard to make the algorithm run in linear time.

Now we turn to the last case $i = 0$, w is the root and it is a pendant.

Lemma 26. *Algorithm 2 deletes a pendant w from G in $O(|N[u]|)$ time when $w = r$.*

Proof. When w is removed, u becomes the next root of the levelwise laminar ordering. Hence, the algorithm has two major maintenance costs; the update of $N(r)$ and $N[r]$ and update of $T(G)$ and $T[G]$. The $N(r)$ and $N[r]$ can be easily updated in $O(|N[u]|)$ time; just replace them by $N(u)$ and $N[u]$, which can be found in $T[G]$ and $T(G)$ in $O(|N[u]|)$ time. Regarding the update of $T(G)$ and $T[G]$, we do not have to remove the sets $N(w)$ and $N[w]$ from $T(G)$ and $T[G]$, since the sets are not represented in the trees. Instead, we have to remove the sets $N(u)$ and $N[u]$ from $T(G)$ and $T[G]$, respectively. This can be done by removing the leaves x in $T[G]$ and $T(G)$ which are pointed to by u, and then removing redundant nodes. The computation time is $O(|N[u]|)$. □

5.3 Linear Construction

Now we are ready to prove the main theorem of this section.

Theorem 27. *Let $G = (V, E)$ be a graph with $n = |V|$ and $m = |E|$. 1) If $G = (V, E)$ is a cograph, the cotree $\mathcal{T}$ derived from G can be constructed in $O(n+m)$ time and space. 2) If G is a distance-hereditary graph, the* DH*-tree $\mathcal{T}$ derived from G can be constructed in $O(n + m)$ time and space.*

Proof. 1) For the cograph, we can straightforwardly use Steps 1∼4, and 6 of the outline given at the beginning of Section 5. We do not need the levelwise laminar ordering, and from Observation 4, Lemma 22, and Theorem 23, two prefix trees can be maintained in $O(|N(w)|)$ time for each removal of w in G.

To get the sets $\mathcal{S}$ and $\mathcal{W}$, we maintain the set of nodes in $T[G]$ and $T(G)$ pointed to by more than one vertex. The vertices pointing to the same node correspond to a set in $\mathcal{S}$ or $\mathcal{W}$. This does not increase either time or space complexity.

2) Now we turn to the distance-hereditary graph. The correctness of the algorithm follows from the results in this section. Hence, we analyze its complexity. The computation of a levelwise laminar ordering takes $O(n+m)$ time and space by Lemma 20. The construction of the prefix trees $T(G)$ and $T[G]$ without $N(r)$ and $N[r]$ requires $O(n+m)$ time and space by Lemma 3. From Lemmas 24∼26, the total time to maintain the prefix trees is bounded by $O(m+n)$.

To complete the proof, we need to show that we can get the sets $\mathcal{S}$, $\mathcal{W}$, and $\mathcal{P}$ in linear time of their sizes, i.e., $O(|\mathcal{S}|)$, $O(|\mathcal{W}|)$, and $O(|\mathcal{P}|)$, respectively. The computation of $\mathcal{P}$ is straightforward. We just maintain the set of vertices of degree one in G, and update it when we remove a vertex. The vertices of degree one are classified by their unique neighbors, and each group classified corresponds to a set P in $\mathcal{P}$.

The computation of $\mathcal{S}$ and $\mathcal{W}$ can be done in the same way as 1) in $O(|\mathcal{S}|)$ and $O(|\mathcal{W}|)$ time, respectively, except for $S^* \in \mathcal{S}$ satisfying $r \in S^*$, and $W^* \in \mathcal{W}$ satisfying $r \in W^*$. To compute them efficiently, we classify the vertices v by $|N(v) \setminus N(r)| - |N(r) \cap N(v)|$ and maintain the value. Only the vertices v satisfying $|N(v) \setminus N(r)| - |N(r) \cap N(v)| = -|N(r)|$ are included in W' or S'. Such a vertex v is in S' if v is adjacent to r, and in W' otherwise.

When we remove a vertex w from G, the value changes only for vertices v satisfying $w \in N(v)$, unless $w = r$. The number of such vertices v is $|N(w)|$; thus, the maintenance can be done in $O(|N(w)|)$ time. When r is removed as a pendant and another vertex becomes the new root, we have to compute $|N(v) \setminus N(r)| - |N(r) \cap N(v)|$ for vertices in $N_1(r)$ and $N_2(r)$. This takes $O(|N[v]|)$ time for each vertex v in $N_1(r) \cup N_2(r)$. Although there will be several root vertex removals, a vertex v is in $N_1(r) \cup N_2(r)$ at most twice. Thus, the total computation time for this operation is bounded by $O(m+n)$.

Therefore, the construction algorithm of the DH-tree $\mathcal{T}$ derived from a given distance-hereditary graph G runs in $O(n+m)$ time and space. □

Corollary 28. *Let $G = (V, E)$ be a graph with $n = |V|$ and $m = |E|$. 1) If $G = (V, E)$ is a cograph, the normalized cotree $\mathcal{T}$ can be constructed in $O(n + m)$ time and space. 2) If G is a distance-hereditary graph, the normalized* DH*-tree $\mathcal{T}$ can be constructed in $O(n + m)$ time and space.*

Proof. By Theorem 27, we can construct the canonical tree derived from G in linear time. We can contract the redundant nodes from the tree in linear time by Lemma 11. □

6 Applications

In this section, we show the applications of the previous sections. Hereafter, we assume that the given graph $G = (V, E)$ is a distance-hereditary graph with n vertices and m edges.

6.1 Recognition and Graph Isomorphism

Theorem 29. 1) *The recognition problem for distance-hereditary graphs can be solved in $O(n + m)$ time and space.* 2) *The graph isomorphism problem for distance-hereditary graphs can be solved in $O(n + m)$ time and space.*

Proof. 1) The modification of the algorithm in Section 5 to determine if the input graph is a distance-hereditary graph is straightforward. First, the algorithm checks if the graph has levelwise laminar ordering. If the vertex sets on some level are not laminar, we reject the graph. Next, the algorithm constructs the open and closed prefix trees $T(G)$ and $T[G]$, and finds the families $\mathcal{S}$, $\mathcal{W}$ and $\mathcal{P}$. Then the algorithm constructs the DH-tree of G step by step. After removing all pendants in $\mathcal{P}$ and contracting all twins in $\mathcal{S} \cup \mathcal{W}$, the algorithm again finds the families $\mathcal{S}$, $\mathcal{W}$ and $\mathcal{P}$. Then, if $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} = \varnothing$ while $V \neq \varnothing$, the algorithm rejects the graph since it is not distance-hereditary graph. It is not difficult to see that this is enough to recognize distance-hereditary graphs by Theorem 2.

2) By Theorem 8 and Corollary 9-1), $G_1 \sim G_2$ iff their corresponding DH-trees are isomorphic as labeled trees. The isomorphism of labeled trees can be checked in linear time (see, e.g., [32, 33]). This together with

Corollary 9-2) completes the proof. □

The characterizations in [18] leads to the following.

Corollary 30. *For cographs and bipartite distance-hereditary graphs, we get the same results as in Theorem* 29.

A cograph is obtained from K_2 by using splitting. In other words, we have no pendant vertices, and hence the levelwise laminar ordering is not required. Hence, our modified algorithm for a cotree is quite simple.

6.2 Enumeration

Let us consider the enumeration problem of distance-hereditary graphs with at most n vertices. More precisely, given n, we efficiently enumerate every distance-hereditary graph with at most n vertices exactly once up to isomorphism. By Lemmas 10 and 15, we have the necessary conditions that a labeled rooted ordered tree is a normalized DH-tree for a distance-hereditary graph. First we show that the conditions are sufficient.

Lemma 31. *Let $\mathcal{T}$ be a labeled rooted ordered tree of diameter d which satisfies all statements from* (a) *to* (e) *in Lemma* 10 *and statements from* (a) *to* (e) *in Lemma* 15. *Then $\mathcal{T}$ is the normalized* DH*-tree of a distance-hereditary $G = (V, E)$ with $|V| > 2$.*

Proof. We prove the statement for the diameter d by induction. We can see the correctness of the lemma for $d \leqslant 2$. Now let us assume that $d > 2$, and the lemma holds for diameters less than d. Let $\mathcal{T}'$ be the tree obtained from $\mathcal{T}$ by peeling, and L be the set of all ordered pairs (x, y) such that x is a leaf removed by the peeling and y is the parent of x which remains in $\mathcal{T}'$. The peeling for $\mathcal{T}$ does not violate the statements in the lemmas. Hence, by the induction hypothesis, $\mathcal{T}'$ is the normalized DH-tree of a distance-hereditary graph G'.

Let G be the graph obtained from G' by the following operations. To each ordered pair (x, y) in L, we apply the operation (α), (β) or (γ) according to the label of y. Then G is a distance-hereditary graph, by definition. If a vertex v in G is generated by one of the above operations, v is in a set in $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$. Thus, by Observation 14, $\mathcal{T}$ is the normalized DH-tree of G. □

Recall that the normalized DH-tree is a canonical form of a distance-hereditary graph G up to isomorphism. That is, for a normalized DH-tree T, there exists a unique distance-hereditary graph and vice versa. Hence, by Lemma 31, it is sufficient to enumerate all labeled rooted ordered tree satisfying the statements in Lemmas 10 and 15 in order to enumerate all distance-hereditary graphs.

Theorem 32. *Distance-hereditary graphs with at most n vertices can be enumerated in $O(n)$ time for each, with $O(n^2)$ space.*

Proof. The enumeration can be done in three steps: 1) enumerate all trees of n leaves such that each inner node has at least two children, with computing center and all isomorphic siblings, 2) for each tree obtained in 1), enumerate all possible assignments of labels to all inner nodes so that the conditions in the characterization are satisfied, and 3) for each label assignment in 2), enumerate all possible choices of one child as a neck for each node with label p.

Note that in 3) we do not distinguish two isomorphic siblings to avoid duplicating same normalized DH-trees. By a slight modification of the tree enumeration algorithm in [34], 1) can be done so that the computation time for each output tree is constant with $O(n)$ space. In a top down depth first manner, both 2) and 3) can be also done in constant time for each with $O(n)$ space. Thus, all normalized DH-trees of distance-hereditary graphs with n vertices can be enumerated in constant time for each. By doing 2) and 3) simultaneously, the average size of the difference between a normalized DH-tree $\mathcal{T}$ and the previously output one $\mathcal{T}'$ is bounded by a constant. Operations $(\alpha), (\beta)$ and (γ) take $O(n)$ time; thus we can construct the distance-hereditary graph whose DH-tree is $\mathcal{T}$ from that of $\mathcal{T}'$ in $O(n)$ time on average. □

6.3 Compact Encoding

In this subsection, we design a simple efficient encoding scheme for distance-hereditary graphs. Our scheme encodes each distance-hereditary graph G with n vertices into only (at most) $4n$ bits in $O(m+n)$ time. One can also decode the string into G in $O(m + n)$ time. After that we design a more efficient encoding scheme which needs only $\lceil 3.5n \rceil$ bits.

Given a distance-hereditary graph with m edges and n vertices one can construct its normalized DH-tree T in $O(m + n)$ time (see Sections 3 and 5). The number of leaf nodes in T is n. Let n_i be the number of inner nodes in T. Since each inner node has two or more children, $n_i \leqslant n - 1$ holds.

We first encode T into a string S_1 with ignoring labels, then encode the labels into a string S_2. The resulting string $S_1 + S_2$ has enough information to reconstruct T and so does G.

Given a normalized (ordered) DH-trees T we traverse T starting at the root and proceed in depth first manner. If we go down an edge of T, we code it with 0, and if we go up an edge, we code it with 1. Thus we need two bits for each edge in T. The length of the resulting bit string is $2(n + n_i - 1) \leqslant 4n - 4$ bits. For

instance, the bit string for the tree in Fig.4 is

$$00001010110001011011101011$$
$$00101000100101011101111.$$

We can save n_i bits from the string above as follows. For each inner node v, after traversing v's first child and its descendant in depth first manner, we return to v again and go "down" to v's second child. Note that each inner node has two or more children. Thus we can omit this "down" to its second child for each inner node. Those $n_i = 10$ bits are underlined in the following bit string:

$$0000\underline{10}1011\underline{00}1\underline{0}11\underline{0}111\underline{0}1011$$
$$\underline{0}01\underline{0}1000\underline{10}01\underline{0}101111\underline{0}111.$$

Thus, we need only $2(n + n_i - 1) - n_i \leqslant 3n - 3$ bits. Let S_1 be the resulting bit string.

We encode the labels of T as follows. Note that each inner node has one label among $\{\mathtt{s}, \mathtt{w}, \mathtt{p}\}$, and each leaf node has no label. We are going to store those labels in preorder with one bit for each label.

Let v be an inner node of T except for the root. Let u be the parent node of v. We show that if the label of u is given, then the label of v has only two choices. By (a) and (b) in Lemma 15, if the label of u is $\mathtt{s}$, then the label of v is not $\mathtt{s}$. Similarly, if the label of u is $\mathtt{w}$, then the label of v is not $\mathtt{w}$. If the label of u is $\mathtt{p}$, we have two subcases. If v is the leftmost child of u, then the label of v is not $\mathtt{p}$; otherwise, the label of v is not $\mathtt{w}$. (Note that two or more neighbors are needed for weak twins.) Thus in any case, the label of node v has only two choices.

Also the label of the root is either $\mathtt{s}$ or $\mathtt{p}$ since we assume that the graph is connected. Thus, we can encode the label of each inner node with only one bit in preorder. The details are as follows.

If the label of the root is $\mathtt{s}$, then we encode it with 0; otherwise, the label is $\mathtt{p}$, and we encode it with 1.

For each inner node v except for the root we have the following three cases. Let u be the parent node of v.

Case 1: the label of u is $\mathtt{s}$. If the label of v is $\mathtt{w}$, we encode it with 0; otherwise, the label is $\mathtt{p}$, and we encode it with 1.

Case 2: the label of u is $\mathtt{w}$. If the label of v is $\mathtt{p}$, we encode it with 0; otherwise, the label is $\mathtt{s}$, and we encode it with 1.

Case 3: the label of u is $\mathtt{p}$. We have two subcases.

(a) v is the leftmost child of u. If the label of v is $\mathtt{s}$, we encode it with 0; otherwise, the label is $\mathtt{w}$, and we encode it with 1.

(b) v is not the leftmost child of u. If the label of v is $\mathtt{s}$, we encode it with 0; otherwise, the label is $\mathtt{p}$, and we encode it with 1.

In this way, we can encode the label of each inner node with only one bit.

By concatenating those bits in preorder, we can encode the labels of inner nodes into a bit string of $n_i \leqslant n - 1$ bits. Let S_2 be the resulting string.

Thus, we have encoded a distance-hereditary graph into a string $S_1 + S_2$ with $2(n + n_i - 1) \leqslant 4n - 4$ bits.

Now we have the following Theorem and Corollary.

Theorem 33. *A distance-hereditary graph $G = (V, E)$ with $|V| = n$ can be represented in $4n$ bits. The number of distance-hereditary graphs of n vertices is at most 2^{4n}.*

Using a simpler case analysis, we also have the following corollary.

Corollary 34. *A cograph $G = (V, E)$ with $|V| = n$ can be represented in $3n$ bits. The number of cographs of n vertices is at most 2^{3n}.*

We can design a more efficient encoding for distance-hereditary graphs. Given a distance-hereditary graph G with m edges and n vertices one can again construct its normalized DH-tree T. We then replace each inner node v with $k \geqslant 3$ children by a binary subtree consisting of $(k - 1)$ new inner nodes $v_1, v_2, \ldots, v_{k-1}$, so that each label of the new inner node is the same label of the original node v. In the resulting tree, each inner node again has one label among $\{\mathtt{s}, \mathtt{w}, \mathtt{p}\}$, each leaf node has no label, and there still is a complete information to reconstruct G. However, now each inner node has exactly two children. By using the level-order binary marked representation in [35], we can encode any binary tree having n leaves into $2n - 1$ bits. After that, we encode the labels of inner nodes in some order using $\log_2 3^{n-1} = (n - 1)\log_2 3$ bits. Thus, we can encode T into $(2 + \log_2 3)n < 3.59n$ bits in total.

Finally, we have the following theorem.

Theorem 35. *The number of distance-hereditary graphs of n vertices is at most $2^{\lceil 3.59n \rceil}$.*

Acknowledgments The authors thank Sheng-Lung Peng for enlightening discussions, sending his paper[28], and pointing out the work about rank-width[27]. The authors also thank Emeric Gioan for fruitful discussions about his recent result[29] at ISAAC 2007.

References

[1] Holder L B, Cook D J, Djoko S. Substructure discovery in the SUBDUE system. In *AAAI Workshop on Knowledge Discovery in Databases*, Seattle, Washington, U.S.A., July 31–August 4, 1994, pp.169–180.

[2] Inokuchi A, Washio T, Motoda H. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. European Conf. Principles and Practice of Knowledge Discovery in Databases* (*PKDD*), Lyon, France, Sept. 13–16, 2000, *LNCS* 1910, Springer-Verlag, pp.13–23.

[3] Zaki M J. Efficiently mining frequent trees in a forest. In *Proc. the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, July 23–26, 2002, ACM Press, pp.71–80.

[4] Asai T, Arimura H, Uno T *et al.* Discovering frequent substructures in large unordered trees. In *Proc. Discovery Science* (*DS'03*), Sapporo, Japan, Oct. 17–19, 2003, *Lecture Notes in Artificial Intelligence*, Vol. 2843, Springer-Verlag, pp.47–61.

[5] Munro J I, Raman V. Succinct representation of balanced parentheses, static trees and planar graphs. In *Proc. the 38th Ann. Symp. Foundation of Computer Science*, Miami Beach, Florida, USA, October 20–22, 1997, ACM Press, pp.118–126.

[6] Munro J I, Raman V. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 2001, 31(3): 762–776.

[7] Nakano S I. Efficient generation of plane trees. *Information Processing Letters*, 2002, 84(3): 167–172.

[8] Geary R, Rahman N, Raman R, Raman V. A simple optimal representation for balanced parentheses. In *Proc. Symp. Combinatorial Pattern Matching* (*CPM*), Istanbul, Turkey, July 5–7, 2004, *LNCS* 3109, Springer-Verlag, pp.159–172.

[9] Knuth D E. Generating All Trees. The Art of Computer Programming, Volume 4, Fascicle 4 Edition, Addison-Wesley, 2005.

[10] Brandstädt A, Le V B, Spinrad J P. Graph Classes: A Survey. SIAM, 1999.

[11] Golumbic M C. Algorithmic Graph Theory and Perfect Graphs. *Annals of Discrete Mathematics*, 57, 2nd Edition, Elsevier, 2004.

[12] Rose D J, Tarjan R E, Lueker G S. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 1976, 5(2): 266–283.

[13] Corneil D G. Lexicographic breadth first search — A survey. In *Proc. Graph-Theoretic Concepts in Computer Science* (*WG 2004*), Bad Honnef, Germany, June 21–23, 2004, *LNCS* 3353, Springer-Verlag, pp.1–19.

[14] Knuth D E. Sorting and Searching. The Art of Computer Programming, Volume 3, 2nd Edition, Addison-Wesley Publishing Company, 1998.

[15] Damiand G, Habib M, Paul C. A simple paradigm for graph recognition: Application to cographs and distance hereditary graphs. *Theoretical Computer Science*, 2001, 263(1/2): 99–111.

[16] Bretscher A, Corneil D, Habib M, Paul C. A simple linear time LexBFS cograph recognition algorithm. In *Proc. Graph-Theoretic Concepts in Computer Science* (*WG 2003*), Elspeet, The Netherlands, June 19–21, 2003, *LNCS* 2880, Springer-Verlag, pp.119–130.

[17] Howorka E. A characterization of distance-gereditary graphs. *Quart. J. Math. Oxford*, 1977, 28(4): 417–420.

[18] Bandelt H-J, Mulder H M. Distance-hereditary graphs. *Journal of Combinatorial Theory, Series B*, 1986, 41(2): 182–208.

[19] D'Atri A, Moscarini M. Distance-hereditary graphs, Steiner trees, and connected domination. *SIAM Journal on Computing*, 1988, 17(3): 521–538.

[20] Hammer P L, Maffray F. Completely separable graphs. *Discrete Applied Mathematics*, 1990, 27(1/2): 85–99.

[21] Chang M S, Hsieh S Y, Chen G-H. Dynamic programming on distance-hereditary graphs. In *Proc. the 8th Int. Symp. Algorithms and Computation* (*ISAAC'97*), Singapore, Dec. 17–19, 1997, *LNCS* 1350, Springer-Verlag, pp.344–353.

[22] Brandstädt A, Dragan F F. A linear-time algorithm for connected r-domination and Steiner tree on distance-hereditary graphs. *Networks*, 1998, 31(3): 177–182.

[23] Broersma H J, Dahlhaus E, Kloks T. A linear time algorithm for minimum fill-in and treewidth for distance hereditary graphs. *Discrete Applied Mathematics*, 2000, 99(1-3): 367–400.

[24] Nicolai F, Szymczak T. Homogeneous sets and domination: A linear time algorithm for distance-hereditary graphs. *Networks*, 2001, 37(3): 117–128.

[25] Hsieh S-Y, Ho C-W, Hsu T-S, Ko M-T. Efficient algorithms for the Hamiltonian problem on distance-hereditary graphs. In *Proc. COCOON 2002*, Singapore, August 15–17, 2002, *LNCS* 2387, Springer-Verlag, pp.77–86.

[26] Chang M S, Wu S-C, Chang G J, Yeh H-G. Domination in distance-hereditary graphs. *Discrete Applied Mathematics*, 2002, 116(1/2): 103–113.

[27] Oum S i. Graphs of bounded rank-width [Ph.D. Dissertation]. Princeton University, 2005.

[28] Chandler D B, Chang M-S, Kloks T, Liu J, Peng S-L. Recognition of probe cographs and partitioned probe distance hereditary graphs. In *Proc. Algorithmic Aspects in Information and Management* (*AAIM*), Hong Kong, China, June 20–22, 2006, *LNCS* 4041, Springer-Verlag, pp.267–278.

[29] Gioan E, Paul C. Dynamic distance hereditary graphs using split decomposition. In *Proc. the 18th International Symposium on Algorithms and Computation* (*ISAAC 2007*), Sendai, Japan, December 17–19, 2007, *LNCS* 4835, Springer-Verlag, pp.41–51.

[30] Spinrad J P. Efficient Graph Representations. American Mathematical Society, 2003.

[31] Corneil D G, Perl Y, Stewart L K. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 1985, 14(4): 926–934.

[32] Lueker G S, Booth K S. A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM*, 1979, 26(2): 183–195.

[33] Colbourn C J, Booth K S. Linear time automorphism algorithms for trees, interval graphs, and planar graphs. *SIAM Journal on Computing*, 1981, 10(1): 203–225.

[34] Nakano S, Uno T. Constant time generation of trees with specified diameter. In *Proc. Graph-Theoretic Concepts in Computer Science* (*WG 2004*), Bad Honnef, Germany, June 21–23, 2004, *LNCS* 3353, Springer-Verlag, pp.33–45.

[35] Jacobson G. Space-efficient static trees and graphs. In *Proc. 30th Symp. Foundations of Computer Science*, North Carolina, October 30–November 1, 1989, *IEEE*, pp.549–554.

Shin-ichi Nakano received the B.E. and M.E. degrees from Tohoku University, Sendai, Japan, in 1985 and 1987, respectively. In 1987 he joined Seiko Epson Corp. and in 1990 he joined Tohoku University. In 1992, he received Ph.D. Eng. degree from Tohoku University. Since 1999 he has been a faculty member of Department of Computer Science, Faculty of Engineering, Gunma University. His research interests are graph algorithms and graph theory. He is a member of IEICE, IPSJ, JSIAM, ACM, and IEEE Computer Society.

Ryuhei Uehara received the 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.

Takeaki Uno received the B.E., M.E., and Ph.D. degrees from Tokyo Institute of Technology, Japan, in 1993, 1995, and 1998, respectively. In 1998∼2001, he was a research associate in Tokyo Institute of Technology, and has been associate professor of National Institute of Informatics, Japan, from 2001. His main research topic is discrete algorithm and its application to industrial/scientific areas. Especially, the topics are enumeration algorithm, graph algorithms, combinatorial optimization, data mining, data engineering, and bioinformatics. He is a member of JSPS, and JORS.