Canonical Tree Representation of Distance Hereditary Graphs with Applications*

Ryuhei Uehara† Takeaki Uno‡

March 6, 2006

Abstract

The class of distance hereditary graphs consists of the isometric graphs. That is, for each pair of vertices, its distance is invariant for any induced path in a distance hereditary graph. In the paper, a canonical tree representation of a distance hereditary graph is proposed. A linear time algorithm for computing the tree representation is also presented. Hence the recognition problem and the graph isomorphism problem for the graph class can be solved in linear time, thanks to linear time isomorphism algorithm for labeled trees. The tree representation takes $O(|V|)$ space for a distance hereditary graph $G = (V, E)$. Hence it can be used as a compact data structure for the graph. It is so informative that all pruning sequences, which is a previously known characterization based on a vertex ordering, can be generated from the tree representation efficiently.

Keywords: algorithmic graph theory, distance hereditary graph, tree representation, recognition problem, isomorphism problem.

1 Introduction

Distance in graphs is one of the most important topics in algorithmic graph theory. The class of distance hereditary graphs was characterized by Howorka to deal with the distance property called *isometric* [13]. More precisely, distance hereditary graphs are graphs in which all induced paths between pairs of vertices have the same length. Some characterizations of distance hereditary graphs are given by Bandelt and Mulder [1], D'Atri and Moscarini [11], and Hammer and Maffray [12]. Especially, Bandelt and Mulder 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 for problems on distance hereditary graphs are based on the characterization [5, 2, 4, 17, 14, 6].

The recognition of a distance hereditary graph based on the the construction of the sequence of extensions can be done in linear time. But it is not so simple (see [10, Chapter 4] for further details); Hammer and Maffray's algorithm [12] fails in some cases, and Damiand, Habib, and Paul's algorithm [10] requires to build the cotree of a cograph in linear time. The cotree of a cograph can be constructed in linear time by using a classic algorithm due to Corneil, Perl, and Stewart [9], or a recent algorithm based on multisweep lexicographically breadth first search (LexBFS) approach by Bretscher, Corneil, Habib, and Paul [3].

On the other hand, while the graph isomorphism problem can be solved in polynomial time, the time bound is not given explicitly for the previously known algorithm (see Spinrad's book [18, p.309]). The recognition algorithm based on the LexBFS approach in [10] fills the vertices into a specific sequence of extensions. Hence this approach seems to be difficult to extend to solve the graph isomorphism problem efficiently. (Similarly, LexBFS approach also allows us to recognize the class of interval graphs quite efficiently [8], but more informative data structure, as PQ-tree, is required to solve the graph isomorphism problem for the class [16].)

*This work was done while the authors were visiting ETH Zürich, Switzerland.

†School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan. uehara@jaist.ac.jp

‡National Institute of Informatics, Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan. uno@nii.jp

In this paper, we first propose a canonical and compact tree representation of a distance hereditary graph. We also give a linear time algorithm that constructs the tree representation of a distance hereditary graph. More precisely, given distance hereditary graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, the algorithm runs in $O(n + m)$ time and space, and the canonical tree representation requires $O(n)$ space. Hence the recognition problem can be solved in $O(n + m)$ time and space. Thanks to linear time isomorphism algorithm for labeled trees, the graph isomorphism problem for distance hereditary graphs can be solved in the same complexity.

The results above give a positive answer to the conjecture that isomorphism can be tested in linear time on the class by reduction to tree isomorphism, due to Spinrad [18, p.309].

The linear time recognition algorithm is not new, but our algorithm is more straightforward than the previously known result which is the combination of results in [10] and [9, 3]. Moreover, our tree representation is so informative that every sequence of extensions can be generated from a tree traverse. Hence it is easy to enumerate all sequences of extensions in a polynomial delay (of the number of vertices and edges).

From the algorithmic point of view, it is worth to mentioning that our algorithm is based on the linear time construction and maintenance of the prefix trees for closed and open neighbors of vertices in a given graph. This is a quite different approach to the previously known algorithms based on (multisweep) LexBFSs [10, 3].

From the graph theoretical point of view, our algorithm uses a new property for a distance hereditary graph. Let $N_i(r)$ denote the set of vertices of distance i from a fixed vertex r. Then, previously known algorithms use the property that the induced subgraphs by $N_i(r)$ are cographs for all i if given graph is a distance hereditary graph. On the other hand, our algorithm uses the laminar structure between $N_i(r)$ and $N_{i+1}(r)$ for each i. This is a generalization of the laminar structure for Ptolemaic graphs, which is shown by Uehara and Uno recently [19]. (A graph is Ptolemaic if and only if it is chordal and distance hereditary.)

We also note that our tree representation is a natural generalization of the notion of the cotree of a cograph. Therefore, omitting the "adding a pendant vertex" extension from our results, we immediately have another simple linear time algorithm for construction of the cotree of a cograph (see [3] for further details of the other linear time algorithms for the class of cographs).

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 by $N(U)$ the set $\{v \in V \mid v \in N(u) \text{ for some } u \in U\}$, and by $N[U]$ the set $\{v \in V \mid v \in N[u] \text{ for some } u \in U\}$. A vertex set C is a *clique* if 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 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 $E' = \{\{u, v\} \mid u, v \in U \text{ and } \{u, v\} \in E\}$. 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 a *twin* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. For a twin u and v, we say that u is a *strong sibling* of v if $\{u, v\} \in E$, and a *weak sibling* if $\{u, v\} \notin E$. We also say *strong (weak) twin* if they are strong (weak) siblings. It is easy to see that strong twin is the transitive relation; that is, if u and v are strong twin and v and w are strong twin, so are u and w. Hence we say that a vertex set S with $|S| > 2$ is also strong twin if each pair in S is strong twin. We also say a vertex set S with $|S| > 2$ is weak twin if each pair in S is weak twin. If a vertex v is a strong or weak twin with another vertex u, we simply say that they are twin, and v has a sibling u. Given a graph $G = (V, E)$, a sequence of the distinct vertices $v_1, v_2, \ldots, v_l$ is a *path*, denoted by $(v_1, v_2, \ldots, v_l)$, if $\{v_j, v_{j+1}\} \in E$ for each $1 \leq j < l$. The *length* of a path is the number of edges on the path. For two vertices u and v, the *distance* of the vertices, denoted by $d(u, v)$, is the minimum length of the paths joining u and v. We here 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.

Given a graph $G = (V, E)$ and a subset U of V, an induced connected subgraph $G[U]$ is *isometric* in G if the distances in $G[U]$ are the same as in G. A graph G is *distance hereditary* if G is connected and every induced path in G is isometric. The following characterization of distance hereditary graphs

is shown by Bandelt and Mulder [1]:

Theorem 1 *A graph G with at least two vertices is distance hereditary if and only if G can be obtained from K_2 by a sequence of extensions of (α) pick any vertex x in G and add x' with an edge $\{x, x'\}$, (β) pick any vertex x in G and add x' with edges $\{x, x'\}$ and $\{x', y\}$ for all $y \in N(x)$, or (γ) pick any vertex x in G and add x' with edges $\{x', y\}$ for all $y \in N(x)$.*

In case (α), we say that the new graph is obtained by attaching a *pendant vertex* x' to the *neck vertex* x. In case (β), x and x' are strong twin, and x and x' are weak twin in case (γ). Hence we say that the new graph is obtained by *splitting* the vertex x into strong twin and weak twin, respectively. In (β) and (γ), the characterization by Bandelt and Mulder assumes that each vertex is split into two siblings. However it is easy to see that we can use the following operations for $k \geq 2$; (β') pick any vertex x in G and split it into k strong siblings, and (γ') pick any vertex x in G and split it into k weak siblings.

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. Then the ordering is *pruning sequence* if 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 twin v and v_i for each $i < n - 1$.

For a vertex set $S \subseteq V$ of $G = (V, E)$, the *contraction* of S (into s) is defined as follows: We first add a new vertex s into V. Then for each edge $\{v, u\}$ with $v \in S$ and $u \in V \setminus S$, add an edge $\{u, s\}$ into E. Multiple edges are replaced by a single edge. Finally, we remove all vertices in S from V and their associated edges from E.

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

2.1 Prefix tree

In this paper, we will maintain $N(v)$ and $N[v] = N(v) \cup \{v\}$ for all vertices in G by two prefix trees. A prefix tree is also called a trie, and the details of the notion can be found in standard textbooks; see, e.g., [15]. A *prefix tree* is a rooted tree with alphabets and labels. To avoid confusion, we call the vertices of prefix trees nodes. Each node except the root has an alphabet, and some nodes have labels. Two or more nodes may have the same alphabet. The alphabets of a prefix tree satisfy the following two conditions; (1) if a node with alphabet a is the parent of a node of alphabet a', it holds $a' > a$, and (2) no two children of a node have the same alphabet. A prefix tree is represented by the set of its nodes, the pointer to the parent from each node, and doubly linked lists of pointers to the children of each node. Thus, we can get the parent of a node in constant time, but to find a specified child of a vertex, such as the child having a given alphabet, takes linear time in the number of children. In our prefix trees, the pointer p from a parent v to the child u is also doubly linked list. That is, from the child u, the pointer p in v that points u can be found in $O(1)$ time. Hence, from u, we can remove the pointer p in v in $O(1)$ time, which will be used for maintaining a pendant vertex.

A path of a prefix tree from a node x to the root gives a set of alphabets, denoted by $set(x)$. If x has a label, then we consider that the label corresponds to $set(x)$. For given alphabet V, a family F of subsets of V can be represented by a prefix tree T: Each set S in F is represented by $set(x)$ for some x in T. In the case, S is a label of the node x. Thus, we can make that any leaf of the prefix tree has at least one label by pruning redundant nodes, and then the size of prefix tree is linear in the total size of the family F.

2.2 Laminar structure of a distance hereditary graph

We here introduce a notation $N_v^+(u)$, $N_v^-(u)$, and $N_v^0(u)$ as follows to show a technical lemma for a distance hereditary graph G. Let v be any vertex in a graph $G = (V, E)$. Then $N_v^+(u)$ denote the set $N_{d(u,v)+1}(v) \cap N(v)$, $N_v^-(u)$ denote the set $N_{d(u,v)-1}(v) \cap N(v)$, and $N_v^0(u)$ denote the set $N_{d(u,v)}(v) \cap N(v)$. We define $N_u^-(u) := \emptyset$.

We note that, for any distance hereditary graph G and r in G, $N_k(r)$ induces a (non connected, in general) cograph, which plays an important role in previous papers [1, 10]. But we use a completely different property.

Let V be a set of n vertices. Two sets X and Y are said to be *overlapping* if $X \cap Y \neq \emptyset$, $X \setminus Y \neq \emptyset$, and $Y \setminus X \neq \emptyset$. A family $\mathcal{F} \subseteq 2^V \setminus \{\{\emptyset\}\}$ is said to be *laminar* if $\mathcal{F}$ contains no overlapping sets; that is, for any pair of two distinct sets X and Y in $\mathcal{F}$ satisfy either $X \cap Y = \emptyset$, $X \subseteq Y$, or $Y \subseteq X$.

Then any distance hereditary graph has the following laminar structure:

Lemma 2 *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 $k \geq 1$.*

Proof. Since $N_r^-(v) = \{r\}$ holds for any $v \in N_1(r) = N(r)$, thus the statement holds for $k = 1$. To show a contradiction for $k > 1$, we assume that there are vertices $u, v \in N_k(r)$ with $k \geq 2$ satisfying that there exist vertices $s \in N_r^-(v) \setminus N_r^-(u)$, $t \in N_r^-(u) \setminus N_r^-(v)$, and $w \in N_r^-(v) \cap N_r^-(u)$.

If v and u are connected by an edge, we consider the graph G' obtained by removing all vertices in $N_r^-(v)$. There is no path connecting r and v in G' such that its length is equal to the distance from r to v in G. On the other hand, there is a path from r to v in G' obtained by attaching u and v to a path from r to t. It contradicts that G is distance hereditary.

If there is no edge connecting v and u, we consider the graph G' obtained by removing all vertices in $N(v) \cap N(u)$. Since there are a path from t to r and a path from s to r, u and v are connected, and the distance from u to v is at least 3. On the other hand, the distance from u to v in G is 2 (by the path (u, w, v)). It contradicts that G is distance hereditary. ∎

We note that the property of Lemma 2 is necessary, but not sufficient condition for the class of distance hereditary graphs. For example, C_6, which is not distance hereditary, also has the property.

2.3 Levelwise laminar ordering of vertices

We here introduce a technical vertex ordering, which plays an important role for removing pendant vertices efficiently. 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. We first order the vertices by the standard breadth first search from r 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 the last part of Q.

Now, we introduce a new notion of an ordering $(v_1 = r, v_2, \ldots, v_n)$ of vertices in V. 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 $0 \leq k < k'$.
(L2) For any $v_i, v_j \in N_k(r), k \geq 1$, $i < j$ holds if $N_r^-(v_j) \subset N_r^-(v_i)$.

Hereafter, we write $v_i < v_j$ for two vertices v_i and v_j if $i < j$.

Lemma 3 *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 4 *For a distance hereditary graph $G = (V, E)$, its levelwise laminar ordering can be found in $O(|V| + |E|)$ time and space.*

Proof. The details of the algorithm is given in Appendix A.1, and omitted here. ∎

We note that the levelwise laminar ordering is a partial order, and it is weaker than lexicographically ordering. Moreover, the linear time algorithm in Appendix A.1 for computing the levelwise laminar ordering for given distance hereditary graph is stable for the breadth first search; ties are broken by the breadth first search manner. Precisely, we assume that the vertex set is given as an ordered set by the levelwise laminar ordering satisfying (L1), (L2), and (L3):

(L3) Let v_i, v_j be any vertices in $N_k(r), k \geq 1, i < j$ 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 $v_i < v < v_j$.

3 DH-tree of a distance hereditary graph

In this section, we introduce the notion of the DH-tree derived from a distance hereditary graph G, which is a canonical and compact representation of 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;

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

We note that when y is the neck of two 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.

Observation 5 *Let P be a set in $\mathcal{P}$. Then 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. ∎

We here show that the families partition V into disjoint sets. More precisely, for any pair of sets S_1 and S_2 in $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$, $S_1 \cap S_2 = \emptyset$. (Note that 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 at least one set. We 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 the unique neck u in P, and by Observation 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 are 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. The similar argument above, there are no other set $P' \in \mathcal{P}$ that contains v. Moreover, the only neighbor of u is v. Hence there are no other vertex v' with $N(v) = N(v')$ and $N[v] = N[v']$ (except $v' = u$, which derive $G \sim K_2$, 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 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 $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$ partitions V into disjoint sets (and the rest vertices). ∎

Here, we introduce the notion of the DH-*tree derived from a distance hereditary graph* G, which is a rooted tree, and each node[1] in the tree has a label which is one of $\{0, +, -, \mathtt{N}, \mathtt{P}\}$. A DH-tree $\mathcal{T}$ contains exactly one node with label 0 which is the root of $\mathcal{T}$. The other labels $+$, $-$, N, P indicate strong twin, weak twin, neck, and pendant, respectively.

For given distance hereditary graph $G = (V, E)$, the DH-tree derived from G is defined recursively. We have three basic cases:

1. The DH-tree derived from K_1 is defined by a single root with label 0.

[1] We will use the notation "node" for a DH-tree to distinguish with a "vertex" in G.

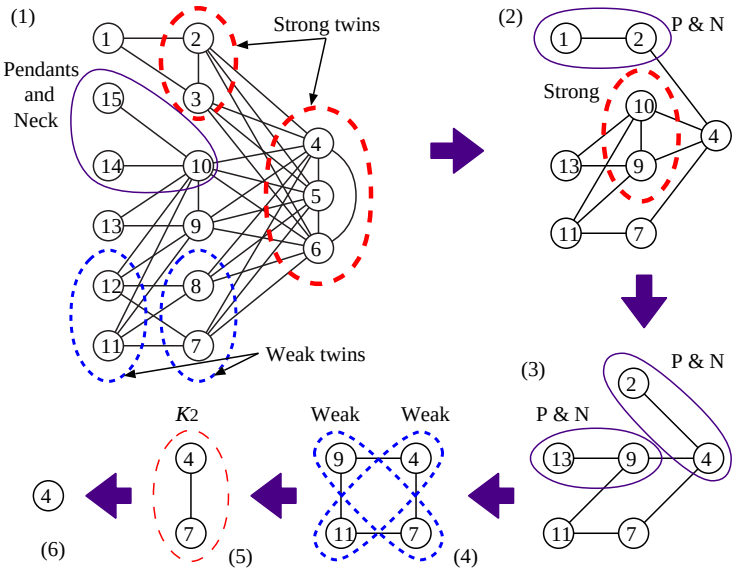

Figure 1: A distance hereditary graph and its contracting/pruning process

2. When $G \sim K_n$ with $n \geq 2$, the DH-tree of G is defined by a single root with label 0 and n children with label $+$. The tree represents that K_n can be obtained from a single vertex K_1 by splitting it into n strong siblings.

3. The graph G is a *star* with $n > 2$ vertices which has a center vertex with $n-1$ pendant vertices. In the case, we define the DH-tree derived from G is defined by a single root with label 0, 1 node with label `N`, and $n-1$ nodes with label `P`.

We note that the number of leaves of the tree is n, which is a property of the DH-tree. We also note that K_2 is not dealt as a star. Hence the root of a DH-tree does not have two children such that one of them is labeled by `N` and the other one is labeled by `P`.

We now define the DH-tree $\mathcal{T}$ derived from $G = (V, E)$ with $|V| = n > 2$ in 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 independent at first. Each leaf in $\mathcal{T}$ corresponds to a vertex in G, and we identify them hereafter. Then, by Lemma 6, we can partition the leaves into the sets in three kinds of families $\mathcal{S}$, $\mathcal{W}$, and $\mathcal{P}$. Let S be any set in $\mathcal{S}$. Then we label the leaves in S by $+$, and make a common parent (with no label) of them. We repeat this process for each S in $\mathcal{S}$. For each set W in $\mathcal{W}$, we similarly label the leaves in W by $-$, and make a common parent. Let P be any set in $\mathcal{P}$. Then, P contains exactly one neck v and some pendants u. We label the neck by `N` and the pendants by `P`, and they have a common parent.

After the process, we contract each vertex set in $\mathcal{S} \cup \mathcal{W}$ into a new vertex on G. Each new vertex corresponds to a parent in the resultant $\mathcal{T}$ and we identify them. We also prune all pendant vertices and leave the neck v in G for each $P \in \mathcal{P}$. The neck v now corresponds to the parent of the nodes in P with $v \in P \in \mathcal{P}$.

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 Fig. 1 and Fig. 2, respectively. The contraction of a twin is described by removing all siblings except the smallest one. Each node in the DH-tree corresponds to a vertex in the distance hereditary graph, and it is depicted in Fig. 2. Note that the DH-tree itself does not store the information.

Theorem 7 *We can construct the* DH*-tree $\mathcal{T}$ derived from G if and only if G is a distance hereditary graph.*

Proof. Due to Bandelt and Mulder in [1], G is distance hereditary if and only if G has a pruning

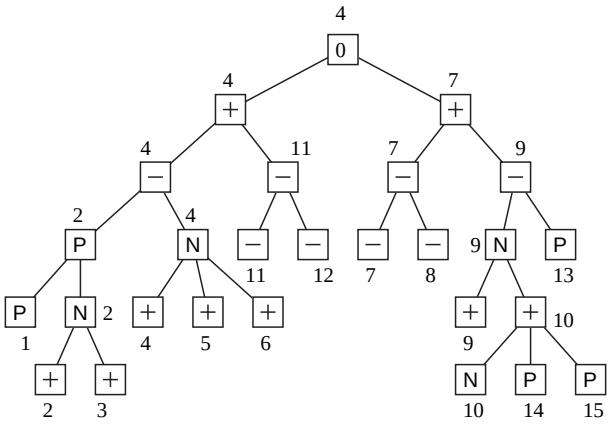

Figure 2: DH-tree $\mathcal{T}$ derived from the graph in Fig. 1(1)

sequence. Hence, using the simple induction for the number of the vertices of G with Lemma 6, we have the theorem. ∎

Corollary 8 *(1) The* DH*-tree derived from a distance hereditary graph is canonical. That is, for any two distance hereditary graphs G_1 and G_2, $G_1 \sim G_2$ if and only if $\mathcal{T}_1 \sim \mathcal{T}_2$ (as labeled trees), where $\mathcal{T}_i$ is the* DH*-tree of G_i for $i = 1, 2$.*

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

Proof. (1) We have the corollary 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 2 children. Thus the number of the inner nodes is at most $|V|$, which implies the corollary. ∎

4 Linear time algorithms for a distance hereditary graph

In this section, we give linear time algorithms for constructing DH-tree of a distance hereditary graph. We first introduce the prefix tree of open and closed neighbors. They are used for detecting the strong twins, weak twins, pendants and necks in short time. We also describe the algorithms to update when we shrink vertices. Hereafter, we assume that $G = (V, E)$ is a given graph with $n = |V|$ vertices and $m = |E|$ edges.

4.1 Open and closed prefix trees and basic operation

In this paper, two families of open neighborhood and closed neighborhood are represented by prefix trees, by considering the neighbor set as an alphabet set. We call them *the open and closed prefix tree*, and denoted by $T(G)$ and $T[G]$, respectively. The prefix trees $T(G)$ and $T[G]$ for the distance hereditary graph G in Fig. 1 are depicted in Fig. 3. Except the root, each square is a node with an alphabet in it. Each circle indicates a vertex v in G, and the thick arrows are pointers to the labels on the node.

Lemma 9 *For any given graph $G = (V, E)$, $T[G]$ and $T(G)$ are constructed in $O(n+m)$ time and space.*

Proof. See Appendix A.2. ∎

From the definitions, we have the following observations.

Observation 10 *Let u and v be any vertices in G. Then*
(1) u is pendant of the neck v if and only if a child of the root of $T(G)$ has alphabet v and label u,
(2) u and v are strong twin if and only if u and v are linked from the same node in $T[G]$, and
(3) u and v are weak twin if and only if u and v are linked from the same node in $T(G)$.

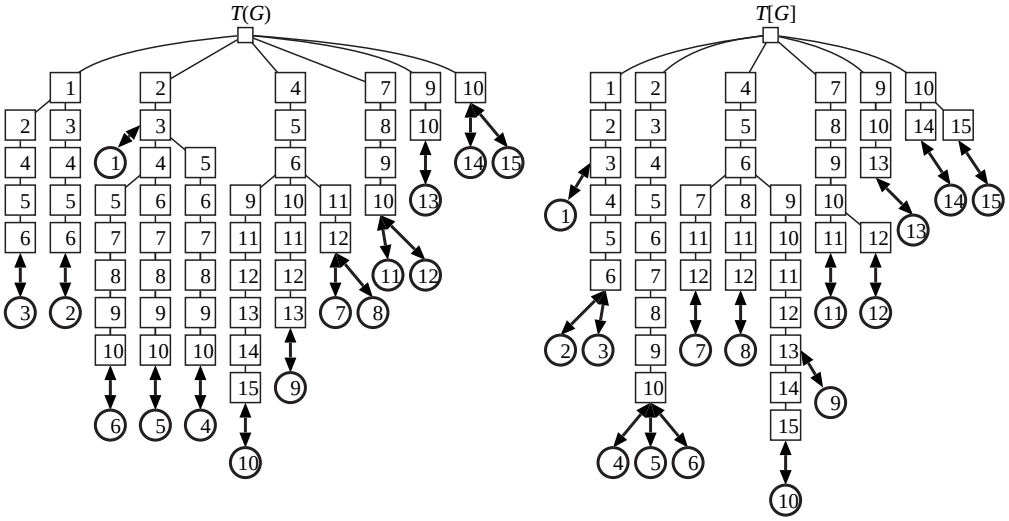

Figure 3: Two prefix trees for G in Fig. 1(1)

From the observation, a set $W \in \mathcal{W}$ (resp., $S \in \mathcal{S}$) corresponds to a set of (two or more) labels on a node in $T(G)$ (resp., $T[G]$), and set $P \in \mathcal{P}$ corresponds to a node with non-empty label of depth 1 in $T(G)$. The outline of the construction of DH-tree $\mathcal{T}$ is as follows;

1. compute a levelwise laminar ordering of V and reorder V according to the ordering (which is crucial when we remove pendants),
2. construct the open and closed prefix trees,
3. if $G \sim K_n$ or G is a star, complete $\mathcal{T}$ and halt,
4. produce nodes of $\mathcal{T}$ for vertices in $\mathcal{P} \cup \mathcal{S} \cup \mathcal{W}$,
5. contract twins and prune pendants with updates of prefix trees $T(G)$ and $T[G]$, and go to step 3.

To perform it in linear time, we have to update prefix trees efficiently. Especially, removing a node from a prefix tree is a key procedure, which is described as follows:

Algorithm 1: Delete a node from a prefix tree

Input : An (open or closed) prefix tree T, node w to be deleted;
Output: Prefix tree T' which does not contain w;

```
1  from the (unique) node x with the label w, delete it;
2  while x is a leaf with no label do delete x and set x to be its parent;
3  foreach node x with alphabet w do
4  |  attach the list of labels on x to the list of the parent y of x;
5  |  connect each child of x to y as a child of y;
6  |  while y has two children z_1, z_2 with same alphabet do
7  |  |  unify them to a node z;
8  |  |  set y := z;
9  |  end
10 |  delete x from T;
11 end
12 return T.
```

Lemma 11 *For a graph $G = (V, E)$ and a vertex w, $T(G - w)$ and $T[G - w]$ are obtained from $T(G)$ and $T[G]$, respectively, by Algorithm 1.* ∎

A node x with a label v has an ancestor with alphabet w if and only if $v \in N(w)$ or $v \in N[w]$. Thus the number of ancestors of any node having label 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 alphabet w is bounded by $|N[w]|$, and the

sum of the number of their children is also bounded by $|N[w]|$. Hence the second loop (from step 3 to step 11) will be repeated at most $|N[w]|$ times. Steps 4 and 10 can be done in $O(1)$ time.

Therefore, our main task is to perform steps 5 and the while loop from step 6 to step 9 efficiently. We here introduce a notion of unification cost as follows: In step 6, for each y, Algorithm 1 finds a pair z_1 and z_2 with same alphabet. Then the *unification cost* for z_1 and z_2 is the time complexity for finding the pair. Then the following lemma is a general property of a prefix tree, which will be used for analysis of deletion of pendants.

Lemma 12 *Suppose that we delete all vertices from $G = (V, E)$ by Algorithm 1. In step 6 of Algorithm 1, for each y, the unification cost for z_1 and z_2 is always $O(1)$, total running time is $O(n + m)$.* ∎

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 twin or pendant. Hereafter, we assume that w is the largest vertex among them. This is not essential for the correctness of the algorithm, but it makes the proofs simpler.

4.2 Prefix tree update for contraction of twins

In this subsection, we assume that w is removed since it is twin. That is, w has a sibling w' with $w' < w$.

Lemma 13 *Let x be any node of prefix tree with alphabet w, and y the parent of x. Then x is the largest node among the children of y.*

Proof. To derive a contradiction, suppose that there is a child x' of y having an alphabet larger than w. Then, no descendant of x' has alphabet w from the definition (1) of prefix tree. 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 with label v, and $N(v)$ contains both of w and w'. Hence, since $w > w'$, there is an ancestor z of x with label w'. Now let x'' be any descendant of x' (including x') with non-empty label v'. Then we have $w' \in N[v']$ and $w \notin N[v']$, which contradicts that w and w' are twin. ∎

By Lemma 13, we can see that no unification occurs in the while loop from step 6 to step 9 in Algorithm 1, since all children of x are larger than any other child of y. Thus, the while loop runs in $O(|N(w)|)$ time (to perform step 5), and hence $T(G - w)$ and $T[G - w]$ are obtained in $O(|N(w)|)$ time.

Theorem 14 *The prefix trees $T(G-w)$ and $T[G-w]$ can be obtained from $T(G)$ and $T[G]$ in $O(|N(w)|)$ time, respectively.* ∎

4.3 Prefix tree update for deletion of pendants

In this subsection, we assume that w is removed since it is a pendant. That is, w has a neck w'. We remind that the vertices are ordered in the levelwise laminar ordering $(r = v_1, v_2, \ldots, v_n)$ from the root r in V.

We start from the special case that $w = r$; in the case, w is the only pendant, since there are no other pendant larger than w.

Observation 15 *Let w' be the unique neighbor of the root pendant r. Then $N(w') \setminus \{r\} = N_2(r)$. Moreover, the levelwise laminar ordering starts with $(v_1 = r = w, v_2 = w', v_3, \ldots, v_k, \ldots)$ such that $\{v_3, \ldots, v_k\} = N_2(r)$ with $k = |N_2(r)| + 2$.*

Lemma 16 *Assume $w = r = v_1$ is the only pendant. We delete v_1 from $T(G)$ and $T[G]$ by Algorithm 1. Then the unification cost in step 6 is always $O(1)$.*

Proof. By Observation 15, v_2 is the neck of v_1. Let x be any node with alphabet v_1 in $T(G)$ or $T[G]$. Then there are three paths containing x with alphabet v_1 in $T(G)$ and $T[G]$, which correspond to $N(v_2)$, $N[v_2]$, and $N[v_1]$.

The path produced by $N(v_2)$ is $(v_1, v_3, v_4, \ldots, v_k)$, where $k = |N_2(r)| + 2$. Thus, when removing v_1 from the path, the remaining path $(v_3, v_4, \ldots, v_k)$ may be unified to the path starts with $(v_3, \ldots)$, which is the third child of the root in $T(G)$. Hence this path can be found in $O(1)$ time. In each level, the path $(v_3, v_4, \ldots, v_k)$ consists of the smallest vertex all possible paths. Hence, in each level, the unification can be checked in $O(1)$ time by checking the left most (or the smallest) child.

The path produced by $N[v_2]$ is $(v_1, v_2, v_3, v_4, \ldots, v_k)$. Hence, similarly, the unification is started from the second child with an alphabet v_2 of the root.

The last path produced by $N[v_1]$ is the path (v_1, v_2) from the root of $T[G]$. Hence its deletion can be done in $O(1)$ time with the first case.

Therefore, in any case, the unification cost in step 6 is always $O(1)$. ∎

We next assume that we are going to remove the pendant w which is not the root (in the case, we can use Lemma 3).

Lemma 17 *Let w be the largest pendant in $N_i(r)$ with $i \geq 1$ (for fixed $\mathcal{P}$). We delete w from $T(G)$ and $T[G]$ by Algorithm 1. Then the unification cost in step 6 is always $O(1)$.*

Proof. Let u be the neck of w, and x be any node with alphabet 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 with alphabet w in $T(G)$ and $T[G]$; they correspond to $N(u)$, $N[u]$, and $N[w]$.

(1) In $T(G)$, x is produced by $N(u)$. We first remark that under x, each node has exactly one child since w has no other neighbor. If w is the maximum vertex in $N(u)$, x is a leaf and it can be removed in $O(1)$ time. Thus we assume that $N(u)$ contains w' with $w' > w$. Since w is the largest pendant in $N_i(r)$ and by Lemma 2, we have $w' \in N_i(r)$, $N_r^-(w') = \{u\}$, and $N_r^0(w') \cup N_r^+(w') \neq \emptyset$. Without loss of generality, we assume that w' is the minimum vertex among the vertices with the property. Then, in $T(G)$, the unique child y of x has the alphabet w'. In $T(G)$, let z be the parent of x.

We first consider the case that z is the root of $T(G)$. Then u has no neighbor w' with $w' < w$. Thus we have either $u = v_1$ or $w = v_1$, where v_1 is the root of the levelwise laminar ordering. By the assumption, $w \neq v_1$ and hence $u = v_1$. Then, by the similar argument in the proof of Lemma 16, the path $(v_3, v_4, \ldots, v_k)$ produced by $N(u)$ (except $w = v_2$) is unified to the leftmost path under the second child of z. Hence the unification cost in step 6 is always $O(1)$.

Hereafter, we assume that z is not the root of $T(G)$, and let v be the alphabet of z. In $T(G)$, let y'_0 be the child of z such that $x < y'_0$ and there are no other child y' of z with $x < y' < y'_0$. That is, y'_0 is the child of z next to x. Hence y'_0 can be found in $O(1)$ time (starting from x), and if y'_0 does not exist, the deletion can be done in $O(1)$ time. Let w'_0 be the alphabet of y'_0. Now, we remind that we are going to remove x with alphabet w, $v < w < w'$, $\{v, w, w'\} \subseteq N(u)$, $w < w'_0$, and there is the minimum vertex u' such that $\{v, w'_0\} \subseteq N(u')$. We have to compare w' and w'_0, and unify them if it is required.

When $w' > w'_0$, we have $w < w'_0 < w'$. Then, if w'_0 is in $N(u)$, it contradicts to the minimality of w'. On the other hand, if w'_0 is not in $N(u)$, it contradicts to (L3). Thus $w' \leq w'_0$. If $w' < w'_0$, we can remove x by replacing x by y; more precisely, each datum in the node x is replaced by the datum in y, which takes $O(1)$ time. The last case is $w' = w'_0$. Then, since $N_r^-(w') = \{u\}$, u' is in $N_i(r) \cup N_{i+1}(r)$. By the definition of the prefix tree, $N(u') \cap \{v_1, v_2, \ldots, v_k = v\} = N(u) \cap \{v_1, v_2, \ldots, v_k = v\}$. Thus, we have $N_r^-(u) = N_r^-(u') = \emptyset$, or consequently, $u = v_1$. Using the same argument to the case that z is the root of $T(G)$, the unification cost in step 6 is always $O(1)$.

(2) In $T[G]$, x is produced by $N[u]$. The case analysis in $T(G)$ works even if $v = u$. Hence the same analysis in $T(G)$ implies the lemma for this case.

(3) In $T[G]$, x is produced by $N[w]$. Since $N[w] = \{w, u\}$ and w is the only neighbor of u, it is easy to see that w is the leaf of the path of length 2. Hence the node x can be removed in $O(1)$ time. ∎

4.4 Levelwise laminar ordering with deletion

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

Lemma 18 *Let $(v_1, v_2, \ldots, v_n)$ be a levelwise laminar ordering of a distance hereditary graph $G = (V, E)$ with the properties (L1), (L2), and (L3). Then $G - v_i$ is a distance hereditary graph, $(v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n)$ is a levelwise laminar ordering of $G - v_i$, and (L1), (L2), and (L3) are satisfied if v_i is either (1) a pendant in a set P in $\mathcal{P}$ of G, or (2) the largest sibling among a twin set S in $\mathcal{S} \cup \mathcal{W}$ of G.*

Proof. By Theorem 1, $G - v_i$ 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 can have some effect to 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 the smallest sibling of v_i. Then $w < v_i$. We first assume that $w, v_i \in N_k(r)$ for some k. Then, removing v_i has effect to the vertices u in $N_{k+1}(r) \cap N(v_i)$. But every set $N_r^-(u)$ containing $\{v_i, w\}$ is replaced by $N_r^-(u) \setminus \{v_i\}$, and its laminar property is not changed since G is distance hereditary. Now, we assume that $w \in N_k(r)$, $v_i \in N_{k'}(r)$, and $k \neq k'$. Since G is distance hereditary, this case occurs only if $w = r = v_1$. Then $w \in N_1(r)$ and $N[w] = N_0(r) \cup N_1(r)$ if $S \in \mathcal{S}$, or $w \in N_2(r)$ and $N(w) = N_1(r)$ if $S \in \mathcal{W}$. In any case, removing v_i has no effect to the ordering. ∎

4.5 Main theorem

Now we are ready to show the main theorem:

Theorem 19 *For any given distance hereditary graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, the* DH*-tree $\mathcal{T}$ derived from G can be constructed in $O(n + m)$ time and space.*

Proof. The correctness of the algorithm follows from Theorem 7 and Lemmas 11 and 18. Hence we analyse its complexity.

The computation of a levelwise laminar ordering takes $O(n + m)$ time and space by Lemma 4. The construction of the prefix trees requires $O(n + m)$ time and space by Lemma 9.

There are two nontrivial bottlenecks we have to consider.

The first bottleneck is the finding the families $\mathcal{S}, \mathcal{W}$ and $\mathcal{P}$ for the current graph. During the update of the prefix trees, especially when update the labels, it it easy to make a set of vertices that are linked from the same node in $T(G)$ and $T[G]$. Hence, by Observation 10(2) and (3), the families $\mathcal{S}$ and $\mathcal{W}$ are easy to find. Maintaining the nodes of depth 2 in $T(G)$, the family $\mathcal{P}$ is also easy to find. Hence the total cost of finding the families can be done in $O(n + m)$ time and space.

The second bottleneck is the total time for the maintenance of prefix trees. By Theorem 14, the deletion of twins requires $O(n + m)$ time in total. By Lemmas 12, 16, and 17, the total cost of deletion of pendants is $O(n + m)$ time and space.

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. ∎

5 Applications

In this section, we assume that given graph $G = (V, E)$ is a distance hereditary graph with n vertices and m edges. We first show the basic problems can be solved in linear time:

Theorem 20 *(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 4 to determine if the input graph is distance hereditary graph requires two steps. First, the algorithm checks if the graph has levelwise laminar ordering. If the vertex sets in some level are not laminar, reject it. Next, the algorithm finds the families $\mathcal{S}, \mathcal{W}$ and $\mathcal{P}$. Then, the input graph is not distance hereditary graph if $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} = \emptyset$, G is not complete graph, and G is not a star. Hence the algorithm rejects in the case. It is not difficult to see that this is enough to recognize distance hereditary graphs by Theorem 1.
(2) By Theorem 7 and Corollary 8(1), $G_1 \sim G_2$ if and only if their corresponding DH-trees are isomorphic as labeled trees. The isomorphism of labeled trees can be checked in linear time. We can see the same usage in [16, 7]. This together with Corollary 8(2) completes the proof. ∎

The DH-tree is compact and informative data structure to represent a distance hereditary graph:

Theorem 21 *For any distance hereditary graph $G = (V, E)$, its pruning sequence can be generated in $O(n + m)$ time and space.*

Proof. Let $\mathcal{T}$ be a given DH-tree of a distance hereditary graph $G = (V, E)$. When G is a star or a complete graph, we have the theorem immediately. Hence we consider the other case. We first regard each leaf of $\mathcal{T}$ as a vertex in V. Then we can construct any pruning sequence as follows:

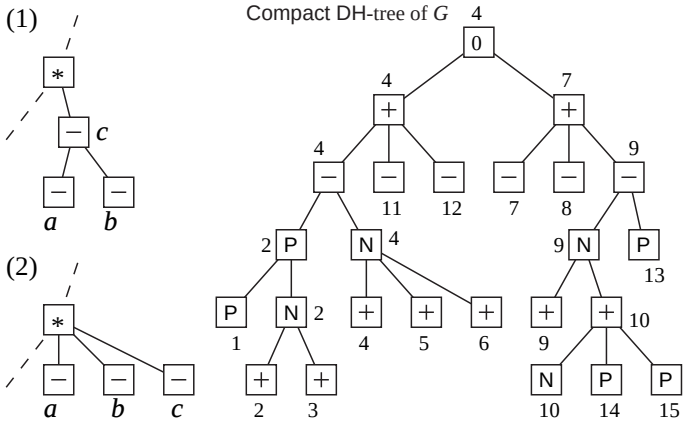

Figure 4: Reduction rule, and the compact DH-tree

Twin contraction: When two nodes x_1 and x_2 with the same label $+$ or $-$ are leaves in $\mathcal{T}$ of the same parent y, we can apply the reverse of the extension (α) or (β). If y has only one leaf x_1, prune x_1 and regard y as the vertex corresponding to x_1.

Pendant pruning: Let x_0 be a node with label `N` in $\mathcal{T}$, and y the parent of x_0. Then, when x_0 is a leaf of $\mathcal{T}$, we can apply the reverse of the extension (γ) to any child x_1 of y with label `P` only if x_1 is also a leaf in $\mathcal{T}$.

It is not difficult to see that the conditions are necessary and sufficient for applying the extensions. A sequence of above operations can be generated using a standard search technique on a tree. Hence, we can compute a pruning sequence in $O(n+m)$ time and space. ∎

Corollary 22 *For any distance hereditary graph $G=(V,E)$, all pruning sequences can be enumerated in polynomial delay and space of n and m, where delay is the maximum computation time between two consecutive output.*

Corollary 23 *For the following graph classes, we have the same results in Theorems 20, 21, and Corollary 22: cographs, bipartite distance hereditary graphs.*

Proof. A graph G with at least two vertices is cograph if and only if G can be obtained from K_2 by the sequence of extensions (β) and (γ) [1]. On the other hand, a graph G with at least two vertices is bipartite distance hereditary graph if and only if G can be obtained from K_2 by the sequence of extensions (α) and (γ) [1]. Hence we have the corollary immediately. ∎

It is worth to remarking that our algorithm modified for a cotree is quite simple, since we have no pendant vertices, and hence the levelwise laminar ordering is not required.

6 Concluding remarks

In Section 3, we introduce the notion of the DH-tree derived from a distance hereditary graph $G=(V,E)$, which is a canonical tree model. However, it can be redundant.

As a rule (β) can be replaced by (β'), if a node with label "$-$" is the parent of the other nodes with label "$-$," they can be weak siblings (see Fig. 4; the case (1) can be replaced by (2)). The same reduction can be applied to the nodes with label "$+$". (A similar reduction can be defined for pendants and necks; when a neck is a parent of the other neck, they can be replaced by a unique neck with associated all pendants. But such redundancy never appear in the DH-tree derived from a distance hereditary graph.)

Hence we can introduce the notion of the *compact* DH-tree of a distance hereditary graph G is obtained from the DH-tree derived from G by applying the reduction as possible as we can. Hence we have the following corollaries for the compact DH-tree.

Corollary 24 *The compact* DH*-tree of a distance hereditary graph is canonical.*

Proof. The ordering of the reductions has no effect on the compact DH-tree. It can be proved by a simple induction of the number of reductions. ∎

Corollary 25 *The compact* DH*-tree of a distance hereditary graph can be constructed from the* DH*-tree derived from the distance hereditary graph in $O(n)$ time and space.*

Proof. From the DH-tree derived from the distance hereditary graph, the compact DH-tree can be constructed by combining the standard depth first search and pointer operations. The algorithm runs in $O(n)$ time, since the DH-tree consist of $O(n)$ nodes. ∎

The compact DH-tree of G in Fig. 1 is depicted in Fig.4. Intuitively, the compact DH-tree implies the following pruning process; the contraction of the vertex sets $\{7, 8\}$ and $\{11, 12\}$ in Fig. 1(2) is postponed until they become $\{7, 8, 9\}$ and $\{4, 11, 12\}$ in Fig. 1(4), respectively. That is, the compact DH-tree gives us the most efficient way to generate the distance hereditary graph from K_2 using the operations (α), (β'), and (γ').

References

[1] H.-J. Bandelt and H.M. Mulder. Distance-Hereditary Graphs. *Journal of Combinatorial Theory, Series B*, 41:182–208, 1986.

[2] A. Brandstädt and F.F. Dragan. A Linear-Time Algorithm for Connected r-Domination and Steiner Tree on Distance-Hereditary Graphs. *Networks*, 31:177–182, 1998.

[3] A. Bretscher, D. Corneil, M. Habib, and C. Paul. A Simple Linear Time LexBFS Cograph Recognition Algorithm. In *Graph-Theoretic Concepts in Computer Science (WG 2003)*, pages 119–130. Lecture Notes in Computer Science Vol. 2880, Springer-Verlag, 2003.

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

[5] M.-S. Chang, S.-Y. Hsieh, and G.-H. Chen. Dynamic Programming on Distance-Hereditary Graphs. In *Proceedings of 8th International Symposium on Algorithms and Computation (ISAAC '97)*, pages 344–353. Lecture Notes in Computer Science Vol. 1350, Springer-Verlag, 1997.

[6] M.-S. Chang, S.-C. Wu, G.J. Chang, and H.-G. Yeh. Domination in distance-hereditary graphs. *Discrete Applied Mathematics*, 116:103–113, 2002.

[7] C.J. Colbourn and K.S. Booth. Linear Time Automorphism Algorithms for Trees, Interval Graphs, and Planar Graphs. *SIAM Journal on Computing*, 10(1):203–225, 1981.

[8] D.G. Corneil, S. Olariu, and L. Stewart. The Ultimate Interval Graph Recognition Algorithm? In *Proc. 9th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 175–180. ACM, 1998.

[9] D.G. Corneil, Y. Perl, and L.K. Stewart. A Linear Recognition Algorithm for Cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.

[10] G. Damiand, M. Habib, and C. Paul. A Simple Paradigm for Graph Recognition: Application to Cographs and Distance Hereditary Graphs. *Theoretical Computer Science*, 263:99–111, 2001.

[11] A. D'Atri and M. Moscarini. Distance-Hereditary Graphs, Steiner Trees, and Connected Domination. *SIAM Journal on Computing*, 17(3):521–538, 1988.

[12] P.L. Hammer and F. Maffray. Completely Separable Graphs. *Discrete Applied Mathematics*, 27:85–99, 1990.

[13] E. Howorka. A Characterization of Distance-Hereditary Graphs. *Quart. J. Math. Oxford (2)*, 28:417–420, 1977.

[14] S.-Y. Hsieh, C.-W. Ho, T.-S. Hsu, and M.-T. Ko. Efficient Algorithms for the Hamiltonian Problem on Distance-Hereditary Graphs. In *COCOON 2002*, pages 77–86. Lecture Notes in Computer Science Vol. 2387, Springer-Verlag, 2002.

[15] D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley Publishing Company, 2nd edition, 1998.

[16] G.S. Lueker and K.S. Booth. A Linear Time Algorithm for Deciding Interval Graph Isomorphism. *Journal of the ACM*, 26(2):183–195, 1979.

[17] F. Nicolai and T. Szymczak. Homogeneous Sets and Domination: A Linear Time Algorithm for Distance-Hereditary Graphs. *Networks*, 37(3):117–128, 2001.

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

[19] R. Uehara and Y. Uno. Laminar Structure of Ptolemaic Graphs and Its Applications. In *16th Annual International Symposium on Algorithms and Computation (ISAAC 2005)*, pages 186–195. Lecture Notes in Computer Science Vol. 3827, Springer-Verlag, 2005.

A Pseudo codes for some algorithms

A.1 The algorithm for computing a levelwise laminar ordering

We here give the complete proof of Lemma 4, which states that a levelwise laminar ordering of a distance hereditary graph $G = (V, E)$ can be computed in $O(|V| + |E|)$ time and space.

First, we fix the root r. Then (L1) is easy to compute using a standard breadth-first search. 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:

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

It is easy to see that, if the problem can be solved in $O(|X| + |Y| + |E'|)$ time and space, a levelwise laminar ordering can be found in linear time. The vertices in X and Y are maintained in usual arrays. To simplify the algorithm, we add a universal vertex y_0 into Y such that $N(y_0) = X$. We first sort $Y' := \{y_0, y_1, \ldots, y_m\}$ such that $|N(y_0)| \geq |N(y_1)| \geq |N(y_2)| \geq \cdots \geq |N(y_n)|$ by bucket sort. (Ties are broken by the original ordering.) Then, if Y can be sorted in a levelwise laminar ordering, the ordering is already a levelwise laminar ordering since $N(y) \subseteq N(y')$ implies $|N(y)| \leq |N(y')|$. Hence, it is sufficient to check if they are not overlapping. To check that, we define the mark(x) for each vertex x in X; mark(x) $= y_j$ indicates that the last vertex set containing x is $N(y_j)$. Hence, if some $N(y_{j'})$ is overlapping $N(y_j)$ with $y_j < y_{j'}$, we can check it since $N(y_{j'})$ contains two vertices x and x' such that mark(x) $\neq$ mark(x'). The details of the algorithm is described below:

Algorithm 2: 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 $E'' = E' \cup \{\{y_0, x_i\} \mid 1 \leq i \leq n\}$;
Output: $Y' \setminus \{y_0\}$;

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

The correctness of Algorithm 2 is easy. It is not difficult to see that Algorithm 2 runs in $O(|X| + |Y'| + |E''|)$ time and space.

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

A.2 The algorithm for construction of a prefix tree

The construction of a prefix tree of a given subset family can be done in linear time. Here we describe the construction algorithm for the open prefix tree as follows.

Algorithm 3: The construction of the open prefix tree

Input : A graph $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$;
Output: $T(G)$;

```
1  // sort neighbors of all vertices by bucket sort;
2  P(v) := ∅ for each v ∈ V;
3  foreach v = 1, 2, ..., n do
4  |  insert v to P(u) for each u ∈ N(v);
5  end
6  // Construct prefix tree level-by-level;
7  T := a tree with one node as root having labels of all vertices of G;
8  foreach k = 0, 1, 2, ... do
9  |  foreach node x on the kth depth of T (in increasing order) do
10 |  |  foreach label v of x (in increasing order) do
11 |  |  |  if (k + 1)st vertex u exists in P(v) then delete label v from x;
12 |  |  |  let y be the child of x having alphabet u;
13 |  |  |  if y does not exist then create y with alphabet u;
14 |  |  |  put label v to y;
15 |  |  end
16 |  end
17 end
18 return T as T(G).
```

The computation time of Algorithm 3 is $O(n+m)$. The first loop is concerned to sort the neighborhood of each vertex at once. This is a kind of bucket sort. In the second loop, we construct the prefix tree level-by-level. In the kth loop, we create all the children of the nodes in the $k-1$st depth. It is done by making a child for each kth vertex in the neighborhood of the vertices of the labels.

Note that the steps 12 and 13 can be done in $O(1)$ time; we put a "mark" y to vertex u in V when the node y is made. All marks of vertices in V for x are cleared after step 15.

The construction of the closed prefix tree can be done in a similar way.