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

Shin-ichi Nakano[1], Ryuhei Uehara[2], and Takeaki Uno[3]

[1] Department of Computer Science, Faculty of Engineering, Gunma University, Gunma 376-8515, Japan
nakano@cs.gunma-u.ac.jp
[2] School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan
uehara@jaist.ac.jp
[3] National Institute of Informatics, Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan
uno@nii.jp

Abstract. Distance-hereditary graphs consist of the isometric graphs, and hence contain trees and cographs. First, a canonical and compact tree representation of the class is proposed. The tree representation can be constructed in linear time using two prefix trees. Usually, the prefix trees are used to maintain a set of strings. The prefix trees in our algorithm are used to maintain the neighbors for each vertex, which is new approach comparing to the other known results based on the lexicographically bread first search. Several efficient algorithms for the distance-hereditary graphs based on the canonical tree representation are proposed; linear time algorithms for graph recognition and graph isomorphism, and efficient enumeration algorithm. An efficient coding for the tree representation is also presented, which requires $4n$ bits for a distance-hereditary graph of n vertices, and $3n$ bits for a cograph. The results improve previously known upper bounds of the number of distance-hereditary graphs and cographs.

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

1 Introduction

Recently, data-driven computations are studied in, e.g., data mining and bioinformatics. We process over tons of data, find knowledge automatically, and classify them in these areas. To achieve the purpose efficiently, we sometimes assume a structure of the data, which is observed implicitly or explicitly. More precisely, we propose a possible structure for the data, enumerate them, and test if the model is feasible. The frequent pattern discovery problem in data mining is a typical example, and widely investigated (see, e.g., [16,19,29,1]). Once the feasible model is fixed, we move our attention to solve the problem over the structured data. However, those structures are relatively primitive from the graph algorithmic point of view, and there are many unsolved problems for more complex structure.

* This work was partially done while the second and third authors were visiting ETH Zürich, Switzerland.

J.-Y. Cai, S.B. Cooper, and H. Zhu (Eds.): TAMC 2007, LNCS 4484, pp. 115–127, 2007.

We have to achieve three efficiencies to deal with the complex structure efficiently; the structure has to be represented efficiently, essentially different instances have to be enumerated efficiently, and the property of the structure has to be checked efficiently. There are few results from this viewpoint except trees [22,23,25,13,20].

Recently, a variety of graph classes have been proposed and studied so far [4,14]. Since an early work by Rose, Tarjan, and Lueker [27], the lexicographic breadth first search (LexBFS) plays an important role as a basic tool to recognize several graph classes. The LexBFS gives us a simple ordering of vertices based on the neighbor-preferred manner (see [9] for further details).

In this paper, instead of LexBFS, we use a prefix tree, which represents a set of strings and supports to check whether a given string is included or not in the set. The notion is also known as trie [21]. We define a string to represent the open and closed neighbors, which will be defined later, and maintain those strings in two prefix trees. Using two prefix trees we can find a set of vertices having identical (or similar) neighbors efficiently. This is a quite different approach to the previously known algorithms based on LexBFS [11,5].

We apply the idea to distance-hereditary graphs. Distance in graphs is one of the most important topics in algorithmic graph theory, and there are many areas that have some geometric property. Distance-hereditary graphs were characterized by Howorka [17] to deal 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 the graph that any pair of vertices u and v has the same distance on any vertex induced subgraph containing u and v. Intuitively, any longer path between them has a shortcut on any vertex induced subgraph.

Some characterizations of distance-hereditary graphs are given [2,12,15]. Especially, Bandelt & Mulder [2] showed that any distance-hereditary graph can be obtained from K_2 by a sequence of operations called "adding a pendant vertex" and "splitting a vertex." Many efficient algorithms on distance-hereditary graphs are based on the characterization [7,3,6,26,18,8]. We will show that the sequence can be found efficiently on the prefix trees that represent open and closed neighbors of each vertex.

In this paper, there are two key contributions for the class of distance-hereditary graphs. First, we propose a compact and canonical tree representation. This is a natural generalization for the tree representation of the class of cographs. Secondly, we show a linear time and space algorithm that constructs the tree representation for any distance-hereditary graph. To achieve the linear time and space, two prefix trees for open and close neighbors play important roles. The results have the following applications.

(1) The graph isomorphism problem can be solved in linear time and space. This is conjectured by Spinrad in [28, p.309], but it was not explicitly given.

(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 [11, Chapter 4] for further details); the original Hammer & Maffray's algorithm [15] fails in some cases, and Damiand, Habib, and Paul's algorithm [11] 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 [10], or a recent algorithm based on multisweep LexBFS approach by Bretscher, Corneil, Habib, and Paul [5].

(3) For 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 $4n$ bits. This encoding gives us a upper bound of the number of distance-hereditary graphs of n vertices: there are at most 2^{4n} 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}. They improve the previously known upper bounds; both are $2^{O(n \log n)}$ [28, p.20,p.98].

Due to space limitation, proofs and some details of algorithms are omitted.

2 Preliminaries

The set of the *open neighbors* 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 neighbors* $N(v) \cup \{v\}$ by $N[v]$. Throughout the paper, we denote by $n := |V|$ and $m := |E|$. For a vertex subset U of V, we denote by $N(U)$ the set $\{v \in V \mid v \in N(u)$ for some $u \in U\}$, and by $N[U]$ the set $\{v \in V \mid v \in N[u]$ 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. 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 $E' = \{\{u, v\} \mid u, v \in U$ 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 *twins* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. For twins 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) twins* if they are strong (weak) siblings. If a vertex v is one of the strong or weak twins with another vertex u, we simply say that they are twins, and v has a sibling u. Since twins make the transitive relation, we say that a vertex set S with $|S| > 2$ is also strong (weak) twins if each pair in S consists of strong (weak) twins. 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 neighbors 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.

The following operations for a graph 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 the 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 the cases, we say that the new graph is obtained by *splitting* the vertex x into strong and weak twins, respectively. It is well known that the class of *cographs* is characterized by the operations as follows:

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).*

The operations are also used by Bandelt and Mulder to characterize the class of *distance-hereditary graphs* [2]:

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)*.*

In (α), a pendant vertex is attached, and in (β) and (γ), a vertex is split into two siblings. We also use the following generalized operations for $k \geq 1$; (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 $G = (V, E)$ and a vertex $s \in S$, the *contraction* of S into s is obtained by: (1) for each edge $\{v, u\}$ with $v \in S \setminus \{s\}$ and $u \in V \setminus S$, add an edge $\{u, s\}$ into E. (2) Multiple edges are replaced by a single edge. (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. Then the ordering is *pruning sequence* if G_{n-1} is K_2 and 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 with only contractions of twins.

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 denote by $G \sim G'$ if they are isomorphic.

2.1 Open and Closed Prefix Trees and Basic Operation

We here introduce the prefix trees, which are also called tries, of open and closed neighbors. The details can be found in standard textbooks; see, e.g., [21]. A *prefix tree* is a rooted tree T; hereafter, the vertices of prefix trees are called *nodes* to distinguish from the vertices in G. Except the root, each node in T is labeled by a vertex in G, and some nodes are linked to vertices in G. We note that two or more nodes in T can have the same label (the name of a vertex of G), but each vertex in G has exactly one pointer to a node in T. The labels of a prefix tree satisfy; (1) if a node with label v is the parent of a node of label v', it holds $v' > v$, and (2) no two children of a node have the same label. We will maintain $N(v)$ and $N[v] = N(v) \cup \{v\}$ for all vertices in G by 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 the node x is pointed by a vertex v, then we consider that $set(x)$ is associated with v. In this manner, two families of open/closed neighbors are represented by two prefix trees, by considering the neighbor set as the associated set. We call them *the open/closed prefix trees*, and denoted by $T(G)$ and $T[G]$, respectively. The prefix trees $T(G)$ and $T[G]$ for the graph G in Fig. 1 are depicted in Fig. 2. Except the root, each square is the node labeled by a vertex in G. Each circle indicates a vertex v in G, and the thick arrows are pointers to the corresponding node. Since we can make that every leaf of the prefix tree is pointed by at least one vertex, the size of prefix tree is $O(n+m)$. With suitable data structure, we can get the parent of a node in constant time, but to find a specified child (by a label) takes linear time in the number of children.

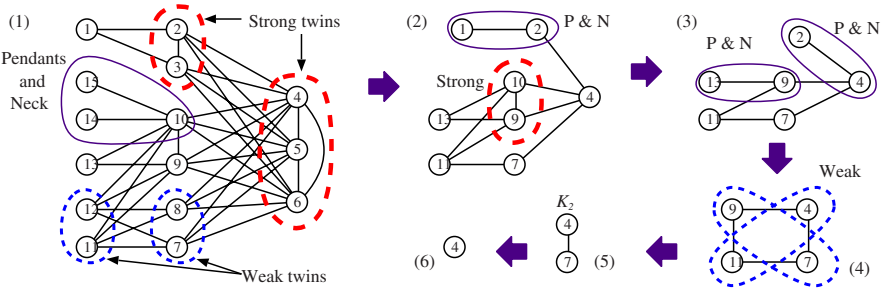

Fig. 1. A distance-hereditary graph and its contracting/pruning process

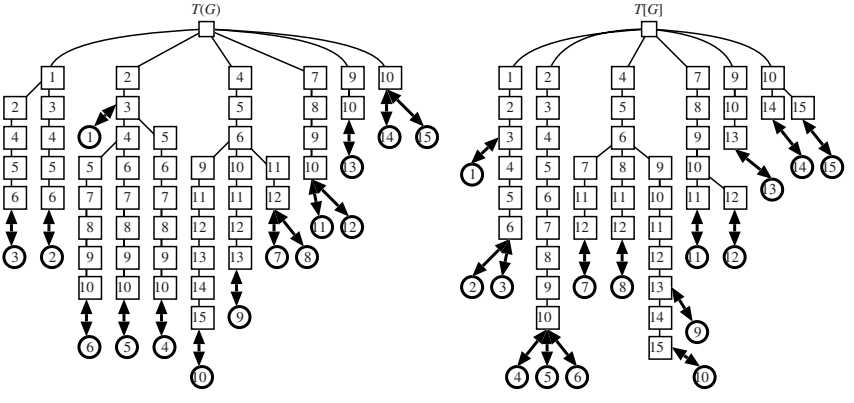

Fig. 2. Two prefix trees for G in Fig. 1(1)

Lemma 1. *For any given graph $G = (V, E)$, $T(G)$ and $T[G]$ are constructed in $O(n+m)$ time and space.*

Observation 3. *Let u and v be any vertices in G. Then (1) u is pendant of the neck v iff a child of the root of $T(G)$ has label v and pointed 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 $T(G)$.*

During the algorithm, we have to update prefix trees efficiently. Especially, removing a node from a prefix tree is a key procedure, which is described in Algorithm 1.

Lemma 2. *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 pointed 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 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 $|N[w]|$, and the sum of the number of their children is also bounded by $|N[w]|$ since a node of label w appears only on the path from the root to the node

Algorithm 1. Delete a vertex from a prefix tree

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

```
1  let x be the node linked to the vertex w; // Remove N[w] in steps 1, 2.
2  while x is a leaf pointed by no vertices do delete x and set x to be its parent;
3  foreach node x with label w do
       // Remove all ws from N(v) or N[v]
4      attach the list of pointers from vertices to 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 z1, z2 with same label do
7          unify z1 and z2 to a node z, and replace z1 and z2 by z;
8          update y by z;
9      delete x from T;
10 return T.
```

pointed by $v \in N[w]$. Hence the loop from steps 3 to 9 will be repeated at most $|N[w]|$ times. Steps 4 and 9 can be done in $O(1)$ time. Therefore, our main task is to perform step 5 and the while loop from step 6 efficiently. We call the step 7 *unification* of z_1 and z_2. The following lemma is a general property of a prefix tree.

Lemma 3. *Suppose that Algorithm 1 deletes all vertices from $G = (V, E)$. Then the total number of unifications is $O(n + m)$.* □

For each y, the *unification cost* of y is time complexity for finding the pair z_1 and z_2. We will show that the unification cost of each y can be $O(1)$.

3 Canonical Trees

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. The derivation is constructive, and the resultant tree is uniquely determined up to isomorphism. But it can be redundant. Hence we next introduce the normalized DH-tree obtained by reducing the redundancy which is also uniquely determined up to isomorphism. Hence it will be used as the canonical and compact representation of a distance-hereditary graph.

We will deal with K_1, K_2 as special distance-hereditary graphs. Hereafter, we assume that $G = (V, E)$ is a connected distance-hereditary graph with at least three vertices. For given G, we define three families of vertex sets as follows;

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

Lemma 4. *(1) Each set $P \in \mathcal{P}$ contains exactly one neck with associated pendants. (2) Each $v \in V$ belongs to either (a) exactly one set in $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$, or (b) no set in the families. (3) For any distance-hereditary graph G, $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \emptyset$.*

We first introduce the notion of the DH-*tree derived from a distance-hereditary graph* G, which is a rooted ordered tree, and each inner node[1] has a label. The label of an inner node is one of $\{\mathtt{s}, \mathtt{w}, \mathtt{p}\}$, which indicate strong/weak twin, and pendant with neck, respectively. The DH-tree derived from G is defined recursively from leaves to the root. We have three basic cases:

(1) The DH-tree derived from K_1 is defined by a single root with no label.

(2) When $G \sim K_n$ with $n \geq 2$, 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) The graph G is a *star* with $n > 2$ vertices which has a center vertex with $n - 1$ pendant vertices. In the case, the DH-tree derived from G is defined by a single root with label p, and n leaves with no labels. We remind that the tree is ordered. In the tree, 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 $n - 1$ leaves correspond to the $n - 1$ pendants.

We note that K_2 is a clique, and not a star. Hence the root with label p of a DH-tree has at least three children. We also note that the number of leaves of the tree is the number of vertices in G. This 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 general case. We assume that G is neither K_n nor a star. We start with independent n leaves of $\mathcal{T}$. Each leaf in $\mathcal{T}$ corresponds to a vertex in G, and we identify them hereafter. Then, by Lemma 4(2), we can group the leaves by three kinds of families $\mathcal{S}$, $\mathcal{W}$, and $\mathcal{P}$. Then, $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \emptyset$ by Lemma 4(3). Let S be any set in $\mathcal{S}$. Then 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 of them. Let P be any set in $\mathcal{P}$. Then, P contains exactly one neck v and some pendants u. We make a common parent with label p of them, and make the neck v the leftmost child of the parent. The pendants are placed on the right of the neck in arbitrary ordering.

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. For each $P \in \mathcal{P}$, we also prune all pendant vertices except the 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 the DH-tree derived from G in Fig. 1 is depicted in Fig. 3. The contraction of twins is described by removing all siblings except the smallest one. Each node in the DH-tree corresponds to a vertex in the graph in Fig. 3.

The DH-tree derived from a distance-hereditary graph can be redundant: As a rule (β) can be replaced by (β'), if a node with label "w" is the parent of the other nodes with label "w," they can be weak siblings (in Fig. 4, the case (1) can be replaced by (2)). The same reduction can be applied to the nodes with label "s".

Hence we can 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 as possible as we can. The reduction can be done by the standard depth

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

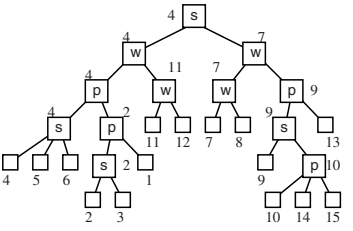

Fig. 3. DH-tree $\mathcal{T}$ derived from the graph in Fig. 1(1)

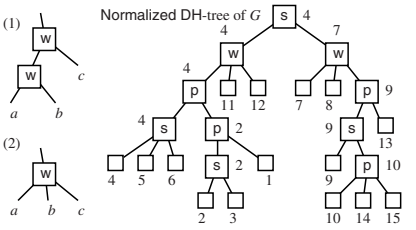

Fig. 4. Reduction rule, and the compact DH-tree

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.

Theorem 4. *The normalized* DH*-tree of a connected distance-hereditary graph is canonical. That is, the normalized* DH*-tree T for a connected distance-hereditary graph G is uniquely determined, and the original distance-hereditary graph G is uniquely constructed from the normalized* DH*-tree T.*

Corollary 1. *The normalized* DH*-tree $\mathcal{T}$ for a distance-hereditary graph $G = (V, E)$ requires $O(|V|)$ space.* □

Note: By Theorems 1 and 2, the normalized DH-tree for a cograph only consists of the nodes of labels s and w. The same notion for a cograph appears in many literature as the *cotree*, e.g., [5,10,11]. Since only the nodes of label p require the ordering of children, cotree of a cograph is the rooted non-ordered tree.

4 Linear Time Construction of Canonical Trees

In this section, we give a linear time algorithm for constructing the DH-tree of a distance-hereditary graph. From Lemma 4(2) and Observation 3, 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 4(1), a set $P \in \mathcal{P}$ contains one neck v. Then, by Lemma 4(1), (2), and Observation 3, the set P corresponds to a node x of depth 1 in $T(G)$ such that (1) x has label v, (2) x is pointed by at least one vertex in G, and (3) all vertex in $P \setminus \{v\}$ points to x. The outline of the construction of the canonical tree for a distance-hereditary graph is; (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) contract twins and prune pendants with updates of prefix trees $T(G)$ and $T[G]$, and go to (2). 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 pendant. Hereafter, we assume that w is the largest vertex among them. We have two cases.

Twin: We first assume that w is the largest twin that has a sibling w' with $w' < w$.

Lemma 5. *Let x be any node of prefix tree with label w, and y the parent of x. Then x is the largest node among the children of y.*

By Lemma 5, no unification occurs in the while loop in Algorithm 1, since all children of x are larger than any other child of y. Thus we have the following:

Theorem 5. *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.*

Pendant: Next we assume that w is a pendant. For the maintenance of a pendant vertex in a distance-hereditary graph, we introduce a technical vertex ordering, called *levelwise laminar ordering* (LL-ordering, for short). We here introduce notations $N_v^+(u)$, $N_v^-(u)$, and $N_v^0(u)$ as follows. Let v be a vertex in $G = (V, E)$. Then $N_v^+(u) := N_{d(u,v)+1}(v) \cap N(v)$, $N_v^-(u) := N_{d(u,v)-1}(v) \cap N(v)$, and $N_v^0(u) := N_{d(u,v)}(v) \cap N(v)$. We define $N_u^-(u) := \emptyset$. Due to space limitation, we only state here the properties of the LL-ordering below:

(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 $N_r^-(v_j) \subset N_r^-(v_i)$.
(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 $v_i < v < v_j$.

The LL-ordering is a partial order weaker than the order by LexBFS. Thus our algorithm runs under weaker assumption than the previous results in [2,11]. If the graph G is a distance-hereditary graph, G has the LL-ordering, and it can be computed in linear time. We also can remove a pendant or contract twins without violating the LL-ordering.

We are ready to remove the largest pendant w. Let w' be the neck of w. Vertices are ordered in the LL-ordering $(r = v_1, v_2, \ldots, v_n)$ from the root r in V. Then we have two subcases. First case is the special case that $w = r$; in the case, w is the only pendant, since there are no other pendant larger than w. Due to lack of space, we omit the case, and we now assume that we aim to remove the pendant w that is not the root.

Lemma 6. *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 is always $O(1)$.*

Theorem 6. *If G is a distance-hereditary graph, the* DH*-tree $\mathcal{T}$ derived from G can be constructed in $O(|V| + |E|)$ time and space.*

5 Applications

Hereafter, we assume that given graph $G = (V, E)$ is a distance-hereditary graph.

Theorem 7. *(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.*

By the characterizations in [2], we have the following:

Corollary 2. *For the following graph classes, we have the same results in Theorem 7: cographs, bipartite distance-hereditary graphs.*

It is worth to remarking that our algorithm modified for a cotree is quite simple, since we have no pendant vertices.

Enumeration: For given n, we efficiently enumerate each distance-hereditary graph with at most n vertices exactly once as follows; (1) enumerate all unordered trees of n leaves such that each inner node has at least two children,(2) for each tree obtained in (1), enumerate all possible assignments of labels to all inner nodes, and (3) for each label assignment in (2), enumerate all possible choices of one child as a neck for each node with label p.Using the technique in [24], we have the following.

Theorem 8. *Distance-hereditary graphs with at most n vertices can be enumerated in $O(n)$ time for each, with $O(n^2)$ space.*

Compact encoding: We design an efficient encoding scheme for distance-hereditary graphs. Our scheme encodes each distance-hereditary graphs G into only (at most) $4n$ bits in $O(m+n)$ time. Also one can decode from the string to G in $O(m+n)$ time.

Given G, one can construct its normalized DH-tree $\mathcal{T}$ in $O(m+n)$ time. The number of leaves in $\mathcal{T}$ is n. Let n_i be the number of inner nodes in $\mathcal{T}$. Since each inner node has two or more children, $n_i \leq n-1$ holds.

We first encode $\mathcal{T}$ 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 $\mathcal{T}$ and so does G.

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

$$000010101100010110111010110010100010010101110111$$

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 with depth first manner, we return back to v again then always 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. In the following bit string those $n_i = 10$ bits are underlined.

$$00001\underline{0}1011\underline{0}001\underline{0}11\underline{0}111\underline{0}1011\underline{0}01\underline{0}10001\underline{0}01\underline{0}1011110111$$

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

Then we encode the labels of $\mathcal{T}$ as follows. Note that each inner node has one label among {s, w, p}, and each leaf 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 $\mathcal{T}$ 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 properties of the DH-tree, if the label of u is s, then the label of v is not s. Similarly, if the label of u is w, then the label of v is not w. If the label of u is p, we have two subcases. If v is the leftmost child of u, then the label of v is not p, otherwise the label of v is not 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 `s` or `p` since we assume that given graph is connected. Thus we can encode the label of each inner node with only one bit in preorder. The detail is as follows.

If the label of the root is `s` then we encode it with 0, otherwise the label is `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 `s`. If the label of v is `w` then we encode it with 0, otherwise the label is `p` and we encode it with 1.

Case 2: The label of u is `w`. If the label of v is `p` then we encode it with 0, otherwise the label is `s` and we encode it with 1.

Case 3: The label of u is `p`. We have two subcases.

Case 3(a): v is the leftmost child of u. If the label of v is `s` then we encode it with 0, otherwise the label is `w` and we encode it with 1.

Case 3(b): v is not the leftmost child of u. If the label of v is `s` then we encode it with 0, otherwise the label is `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 $n_i \leq n - 1$ bits. Let S_2 be the resulting string.

Thus we have encoded a distance-hereditary graph into a string $S_1 + S_2$ with $2(n + n_i - 1) \leq 4n - 4$ bits. Now we have the following Theorem and Corollary.

Theorem 9. *A distance-hereditary graph G can be represented in $4n$ bits.*

Corollary 3. *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 4. *A cograph G can be represented in $3n$ bits, and the number of cographs of n vertices is at most 2^{3n}.*

References

1. T. Asai, H. Arimura, T. Uno, and S. Nakano. Discovering Frequent Substructures in Large Unordered Trees. In *Discovery Science (DS '03)*, pages 47–61. Lecture Notes in Artificial Intelligence Vol. 2843, Springer-Verlag, 2003.
2. H.-J. Bandelt and H.M. Mulder. Distance-Hereditary Graphs. *Journal of Combinatorial Theory, Series B*, 41:182–208, 1986.
3. 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.
4. A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.
5. 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.

6. 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.
7. 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.
8. 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.
9. D.G. Corneil. Lexicographic Breadth First Search — A Survey. In *Graph-Theoretic Concepts in Computer Science (WG 2004)*, pages 1–19. Lecture Notes in Computer Science Vol. 3353, Springer-Verlag, 2004.
10. D.G. Corneil, Y. Perl, and L.K. Stewart. A Linear Recognition Algorithm for Cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985.
11. 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.
12. A. D'Atri and M. Moscarini. Distance-Hereditary Graphs, Steiner Trees, and Connected Domination. *SIAM Journal on Computing*, 17(3):521–538, 1988.
13. R. Geary, N. Rahman, R. Raman, and V. Raman. A Simple Optimal Representation for Balanced Parentheses. In *Symposium on Combinatorial Pattern Matching (CPM)*, pages 159–172. Lecture Notes in Computer Science Vol. 3109, Springer-Verlag, 2004.
14. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics 57. Elsevier, 2nd edition, 2004.
15. P.L. Hammer and F. Maffray. Completely Separable Graphs. *Discrete Applied Mathematics*, 27:85–99, 1990.
16. L. B. Holder, D. J. Cook, and S. Djoko. Substructure Discovery in the SUBDUE System. In *Workshop on Knowledge Discovery in Databases*, pages 169–180. AAAI Workshop on Knowledge Discovery in Databases, AAAI, 1994.
17. E. Howorka. A Characterization of Distance-Hereditary Graphs. *Quart. J. Math. Oxford (2)*, 28:417–420, 1977.
18. 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.
19. A. Inokuchi, T. Washio, and H. Motoda. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In *European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 13–23. Lecture Notes in Computer Science Vol. 1910, Springer-Verlag, 2000.
20. D. E. Knuth. *Generating All Trees*, volume 4 of *The Art of Computer Programming*. Addison-Wesley, fascicle 4 edition, 2005.
21. D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley Publishing Company, 2nd edition, 1998.
22. J. I. Munro and V. Raman. Succinct Representation of Balanced Parentheses, Static Trees and Planar graphs. In *Proc. 38th ACM Symp. on the Theory of Computing*, pages 118–126. ACM, 1997.
23. J. I. Munro and V. Raman. Succinct Representation of Balanced Parentheses and Static Trees. *SIAM Journal on Computing*, 31:762–776, 2001.
24. S. Nakano and T. Uno. Constant Time Generation of Trees with Specified Diameter. In *Graph-Theoretic Concepts in Computer Science (WG 2004)*, pages 33–45. Lecture Notes in Computer Science Vol. 3353, Springer-Verlag, 2005.
25. S.-I. Nakano. Efficient Generation of Plane Trees. *Information Processing Letters*, 84(3):167–172, 2002.

26. F. Nicolai and T. Szymczak. Homogeneous Sets and Domination: A Linear Time Algorithm for Distance-Hereditary Graphs. *Networks*, 37(3):117–128, 2001.
27. D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.
28. J.P. Spinrad. *Efficient Graph Representations*. American Mathematical Society, 2003.
29. M. J. Zaki. Efficiently Mining Frequent Trees in a Forest. In *8th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, ACM Press, 2002.