SIAM J. DISCRETE MATH.
Vol. 15, No. 4, pp. 488–518

CHARACTERIZATION OF EFFICIENTLY PARALLEL SOLVABLE PROBLEMS ON DISTANCE-HEREDITARY GRAPHS*

SUN-YUAN HSIEH†, CHIN-WEN HO‡, TSAN-SHENG HSU§, MING-TAT KO§, AND GEN-HUEY CHEN¶

Abstract. In this paper, we sketch common properties of a class of so-called subgraph optimization problems that can be systematically solved on distance-hereditary graphs. Based on the found properties, we then develop a general problem-solving paradigm that solves these problems efficiently in parallel. As a by-product, we also obtain new linear-time algorithms by a sequential simulation of our parallel algorithms. Let $T_d(|V|,|E|)$ and $P_d(|V|,|E|)$ denote the time and processor complexities, respectively, required to construct a decomposition tree of a distance-hereditary graph $G=(V,E)$ on a PRAM model M_d. Based on the proposed paradigm, we show that the maximum independent set problem, the maximum clique problem, the vertex connectivity problem, the domination problem, and the independent domination problem can be sequentially solved in $O(|V|+|E|)$ time, and solved in parallel in $O(T_d(|V|,|E|)+\log|V|)$ time using $O(P_d(|V|,|E|)+|V|/\log|V|)$ processors on M_d. By constructing a decomposition tree under a CREW PRAM, we also show that $T_d(|V|,|E|)=O(\log^2|V|)$ and $P_d(|V|,|E|)=O(|V|+|E|)$.

Key words. algorithms, data structures, distance-hereditary graphs, parallel random access machine, subgraph optimization problems

AMS subject classifications. 68P05, 68Q25, 68R10, 68W10, 68W40

PII. S0895480101389880

1. Introduction.

A graph is *distance-hereditary* [2, 18] if the distance stays the same between any of two vertices in every connected induced subgraph containing both (where the *distance* between two vertices is the length of a shortest path connecting them). Distance-hereditary graphs form a subclass of perfect graphs [11, 15, 18] that are graphs G in which the maximum clique size equals the chromatic number for every induced subgraph of G [3, 13]. Two well-known classes of graphs, trees and cographs, both belong to distance-hereditary graphs. There were sequential or parallel algorithms to solve quite a few interesting graph-theoretical problems on this special class of graphs. The interested readers may consult [2, 5, 6, 11, 12, 15, 16, 18, 19, 20, 25, 27, 28, 29] for details.

*Received by the editors May 25, 2001; accepted for publication (in revised form) June 20, 2002; published electronically September 10, 2002. Two various parts of this paper appeared in *Proceedings of the 9th International Symposium on Algorithms and Computation (ISAAC'98), Lecture Notes in Comput. Sci.* 1533, 1998, *pp.* 257–266, and in *Proceedings of the 4th International ACPC Conference Including Special Tracks on Parallel Numerics (ParNum'99) and Parallel Computing in Image Processing, Video Processing, and Multimedia (ACPC'99), Lecture Notes in Comput. Sci.* 1557, 1999, *pp.* 417–426.

http://www.siam.org/journals/sidma/15-4/38988.html

†Department of Computer Science and Information Engineering, National Cheng Kung University, 1 University Rd., Tainan 701, Taiwan (hsiehsy@mail.ncku.edu.tw). The work of this author was supported in part by the National Science Council under grant NSC 89-2218-E-260-015 and by the Institute of Information Science, Academia Sinica, Taiwan. Part of this work was done while this author was visiting Academia Sinica, Taiwan.

‡Department of Computer Science and Information Engineering, National Central University, Chung-Li, Taiwan (hocw@csie.ncu.edu.tw). The work of this author was supported in part by the National Science Council under grant NSC 88-2213-E-008-001.

§Institute of Information Science, Academia Sinica, Taipei, Taiwan (tshsu@iis.sinica.edu.tw, mtko@iis.sinica.edu.tw).

¶Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan (ghchen@csie.ntu.edu.tw).

Several characterizations of distance-hereditary graphs were also explored for algorithmic applications. In [2], Bandelt and Mulder showed that the house, holes, domino, and gem are neither induced subgraphs nor isometric subgraphs of a distance-hereditary graph. In [15], Hammer and Maffray utilized the hanging structure to show that a graph is distance-hereditary if and only if it has a one-vertex-extension ordering. Using this ordering, they proposed a linear $O(|V| + |E|)$-sequential-time recognition algorithm, where V and E are the vertex and edge sets of the given graph. The vertex-coloring problem and the maximum weighted stable set problem were also solved in linear time in [15]. In [6], Chang, Hsieh, and Chen generalized the concept of the one-vertex-extension ordering to define the one-vertex-extension tree. They further obtained a new recursive definition of distance-hereditary graphs and showed that this new characterization can be utilized to solve the weighted vertex cover problem, the weighted independent domination problem, the minimum fill-in problem, and the tree-width problem. The former (respectively, latter) two problems need $O(|V| + |E|)$ (respectively, $O(|V|^2)$) sequential time. Quite recently, Golumbic and Rotics [14] showed that distance-hereditary graphs are those graphs of clique-width at most three for which a corresponding 3-expression can be built in linear sequential time. Moreover, Courcelle, Makowsky, and Rotics [9] showed an elegant result that given a k-expression of a graph G with the bounded clique-width k, all graph problems expressible in monadic second order logic with quantification over vertex sets only can be solved in linear time on G. Therefore, a wide class of graph problems are linear-time solvable on distance-hereditary graphs.

Most known polynomial time algorithms on distance-hereditary graphs utilize techniques discovered from the properties of the problems and graphs, which we feel are inherently sequential. In this paper, we propose a new approach based on the one-vertex-extension tree proposed in [6] to come out a general problem-solving paradigm, and thus a good structure for representing distance-hereditary graphs, for designing parallel algorithms for a class of problems on distance-hereditary graphs. Note that we also obtain linear-time algorithms that are different from the previous studies of other researchers for all these problems by sequentially simulating our parallel algorithms. Given a graph problem, we say it belongs to the class of *subgraph optimization* problem if the object of this problem is to find a subgraph of the input graph to satisfy the given properties which include an optimization constraint. For example, the problem of finding a maximum independent set is a subgraph optimization problem. By discovering recursive properties of distance-hereditary graphs, we define a general problem-solving paradigm for subgraph optimization problems. The paradigm consists of the two main phases. The first phase is to construct a binary tree structure, called a *decomposition tree*, for representing a distance-hereditary graph. The second phase is to reduce the given subgraph optimization problem to another problem which can be solved on a decomposition tree. Problems that fit in our paradigm include the following: (a) the maximum clique problem, (b) the maximum independent set problem, (c) the vertex connectivity problem, (d) the domination problem, and (e) the independent domination problem. All the above problems but problem (c) were shown to be linear-time solvable [6, 9, 14, 15].

Let $T_d(|V|, |E|)$ and $P_d(|V|, |E|)$ denote the time complexity and processor complexity required to construct a decomposition tree of a distance-hereditary graph $G = (V, E)$ on a PRAM model M_d. We show that problems (a)–(e) can be sequentially solved in $O(|V| + |E|)$ time, and solved in parallel in $O(T_d(|V|, |E|) + \log |V|)$ time using $O(P_d(|V|, |E|) + |V|/\log |V|)$ processors on M_d. If a decomposition tree

is given to be the input instance, problems (a)–(e) can be solved in $O(\log |V|)$ time using $O(|V|/\log |V|)$ processors on an EREW PRAM. To our knowledge, the sequential complexity of problem (c) and the parallel complexities of problems (b)–(e) remains unknown in the literatures. Note that previous known parallel complexities for problem (a) on distance-hereditary graphs were $O(\log^2 |V|)$ time using $O(|V|+|E|)$ processors on a CREW PRAM [19]. For the rest, we match the current best algorithms [6, 15, 19]. By constructing a decomposition tree in parallel, we also show that $T_d(|V|, |E|) = O(\log^2 |V|)$, $P_d(|V|, |E|) = O(|V| + |E|)$ under a CREW PRAM.

The computation model used here is the deterministic parallel random access machine (PRAM) which permits concurrent read and exclusive write (CREW), or exclusive read and write (EREW) in its shared memory [22]. The rest of this paper is organized as follows. In section 2, we review some properties of distance-hereditary graphs and give basic definitions. In section 3, we define a general problem-solving paradigm and develop its sequential and parallel implementation. In section 4, we show that problems (a)–(e) are examples that fit into our paradigm. In section 5, we present a parallel algorithm to construct a decomposition tree for a distance-hereditary graph. Finally, some concluding remarks are given in section 6.

2. Preliminaries. This paper considers finite, simple, and undirected graphs $G = (V, E)$, where V and E are the vertex and edge sets of G, respectively. Let $n = |V|$ and $m = |E|$. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, the *union of* G_1 *and* G_2, denoted by $G_1 \cup G_2$, is the graph $(V_1 \cup V_2, E_1 \cup E_2)$. Let $G[X]$ denote the subgraph of G induced by $X \subseteq V$. For graph-theoretic terminologies and notations not mentioned here, see [13]. For a vertex $v \in V$ of a graph $G = (V, E)$, the *neighborhood* of v is $N_G(v) = \{u \in V |\ (u, v) \in E\}$ and the *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. We use $N(v)$ for $N_G(v)$, and $N[v]$ for $N_G[v]$, if there is no ambiguity.

For a graph $G = (V, E)$, the *degree* of a vertex $v \in V$ is $deg(v) = |N(v)|$. We say that vertex u is a *pendant vertex* attached to vertex v if $deg(u) = 1$ and v is the vertex adjacent to u. Two vertices u and v are called *true* (respectively, *false*) *twins* if $N[u] = N[v]$ (respectively, $N(u) = N(v)$).

Given a graph $G = (V, E)$, an ordering $\delta = (v_1, v_2, \ldots, v_n)$ of V is said to be a *one-vertex-extension ordering* of G if v_i is a pendant vertex attached to some vertex in $G[V_i]$ or is a twin of some vertex in $G[V_i]$ for $1 \le i \le n$, where $V_i = \{v_1, v_2, \ldots, v_i\}$.

LEMMA 2.1 (see [2, 15]). *A graph is distance-hereditary if and only if it has a one-vertex-extension ordering.*

Let $G = (V, E)$ be a distance-hereditary graph with a one-vertex-extension-ordering $\delta = (v_1, v_2, \ldots, v_n)$. In [6], Chang, Hsieh, and Chen constructed a *one-vertex-extension tree*, denoted by $\mathcal{E}_G$, with respect to δ as follows. Tree $\mathcal{E}_G$ is a rooted ordered tree rooted at v_1 with the node set V. For $j = 2, 3, \ldots, n$, we let v_j be the rightmost child of v_i, $i < j$, in the current tree if either v_j is a pendant vertex attached to v_i or v_j and v_i are twins in $G[V_j]$. We use (v_j, v_i) to denote an edge of $\mathcal{E}_G$. Moreover, (v_j, v_i) is labelled with P if v_j is a pendant vertex attached to v_i in $G[V_j]$, and it is labelled with T (respectively, F) if v_i and v_j are true twins (respectively, false twins) in $G[V_j]$.

LEMMA 2.2 (see [6]). *A one-vertex-extension tree of a distance-hereditary graph can be constructed in $O(n + m)$ time.*

Figure 2.1(a) shows a distance-hereditary graph whose vertex set is associated with a one-vertex-extension ordering. Figure 2.1(b) shows a one-vertex-extension tree with respect to the ordering.

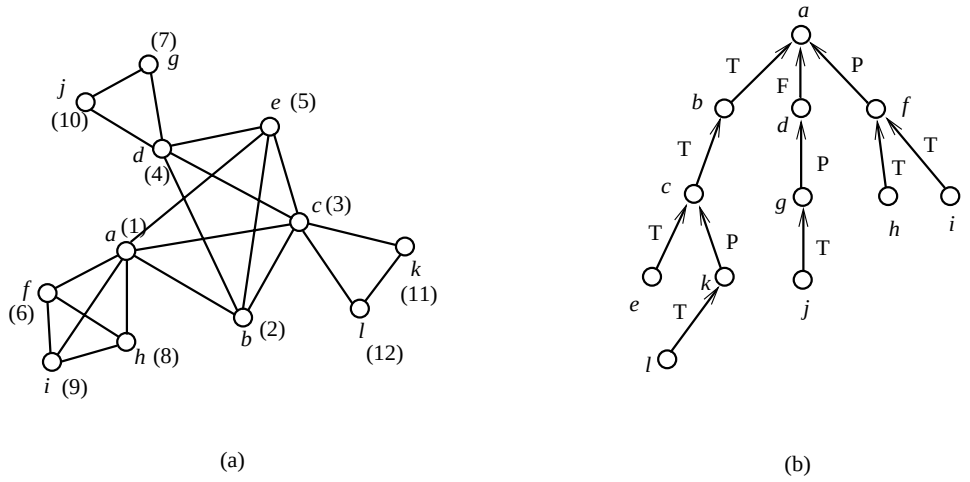

FIG. 2.1. *A distance-hereditary graph and its one-vertex-extension tree. The numbers* (1)–(12) *associated with the vertices of the graph shown in* (a) *correspond to a one-vertex-extension ordering.*

The *twin set of* $v \in V(\mathcal{E}_G)$, denoted by $S(v)$, consists of v and the descendants of v such that v can be reached through only T or F edges. The *twin set of* $\mathcal{E}_G$ (or *twin set of* G) is the twin set of the root of $\mathcal{E}_G$. In Figure 2.1(b), the twin set of G is $\{a, b, c, d, e\}$.

Suppose nodes $v_{i_1} < v_{i_2} < \cdots < v_{i_{j-1}} < v_{i_j} < v_{i_{j+1}} < \cdots < v_{i_k}$ are children of v_i in $\mathcal{E}_G$. For an edge (v_{i_j}, v_i) in $\mathcal{E}_G$, let $S_r(v_{i_j}, v_i) = S(v_i) \setminus (\cup_{l=1}^{j} S(v_{i_l}))$. Let $\mathcal{E}_G(v_{i_j}, v_i)$ denote the subtree of $\mathcal{E}_G$ induced by $v_i, v_{i_j}, v_{i_{j+1}}, \ldots, v_{i_k}$ and all descendants of $v_{i_j}, v_{i_{j+1}}, \ldots, v_{i_k}$. Recall that $\mathcal{E}_G(v)$ is used to denote the subtree rooted at v in $\mathcal{E}_G$.

We say that two disjoint vertex subsets X and Y *form a join* in a graph $G = (V, E)$ if every vertex of X is connected to every vertex of Y.

LEMMA 2.3 (see [6]). *Suppose that v_j is a child of v_i in $\mathcal{E}_G$. Then the following two statements hold.*

1. *If (v_j, v_i) is labelled with P or T, then $S(v_j)$ and $S_r(v_j, v_i)$ form a join in G. Moreover, for every vertex $v \in V(\mathcal{E}_G(v_j)) \setminus S(v_j)$, $N[v] \subseteq V(\mathcal{E}_G(v_j))$.*

2. *If (v_j, v_i) is labelled with F, then every vertex of $V(\mathcal{E}_G(v_j))$ is not adjacent to any vertex of $V(\mathcal{E}_G(v_j, v_i)) \setminus V(\mathcal{E}_G(v_j))$ in G.*

Given a distance-hereditary graph $G = (V, E)$, there exists a one-vertex-extension ordering $(v_1, v_2, \ldots, v_n)$. This ordering corresponds to a one-vertex-extension tree $\mathcal{E}_G$. Note that the twin set of G is $S(v_1)$. The vertex set V can be partitioned into four disjoint sets: $V(\mathcal{E}_G(v_2)) \setminus S(v_2)$, $S(v_2)$, $(V \setminus V(\mathcal{E}_G(v_2))) \setminus S_r(v_2, v_1)$, and $S_r(v_2, v_1)$. By Lemma 2.3, G can be regarded as to be formed from $G_1 = G[V \setminus V(\mathcal{E}_G(v_2))]$ and $G_2 = G[V(\mathcal{E}_G(v_2))]$ by the three operations according to the type (v_2, v_1) in $\mathcal{E}_G$ as follows. If (v_2, v_1) is labelled T or P, then G is formed from G_1 and G_2 by connecting every vertex of $S_r(v_2, v_1)$ to all vertices of $S(v_2)$. If (v_2, v_1) is labelled F, then G is the union of G_1 and G_2. If (v_2, v_1) is labelled P, then the twin set of G is the twin set of G_1. If $[v_1, v_2]$ is labelled T or F, then the twin set of G is the union of the twin set of G_1 and G_2. Based upon the above observations, we provide a characterization for distance-hereditary graphs below.

A graph consisting of a single vertex v is clearly a distance-hereditary graph. It is said to be a *primitive distance-hereditary graph with the twin set* $\{v\}$ [6]. A graph G with $|V(G)| \geq 2$ is distance-hereditary if and only if it can be obtained by three operations described in the following lemma. Let G_1 and G_2 be distance-hereditary graphs with the twin sets S_1 and S_2, respectively.

LEMMA 2.4 (see [6]). 1. *The graph obtained from G_1 and G_2 by connecting every vertex of S_1 to all vertices of S_2 is a distance-hereditary graph with the twin set $S_1 \cup S_2$.*

2. *The graph obtained from G_1 and G_2 by connecting every vertex of S_1 to all vertices of S_2 is a distance-hereditary graph with the twin set S_1.*

3. *The union of G_1 and G_2 is a distance-hereditary graph with the twin set $S_1 \cup S_2$.*

Note that the difference between operations 1 and 2 of Lemma 2.4 is the twin set construction.

A distance-hereditary graph G is said to be formed from G_1 with the twin set S_1 and G_2 with the twin set S_2 by the *true twin* (respectively, *attachment*) operation if G is obtained through operation 1 (respectively, 2) of Lemma 2.4, and by the *false twin operation* if G is obtained through operation 3 of Lemma 2.4.

A distance-hereditary graph can be represented by a binary tree form, called a *decomposition tree*, which is defined as follows.

DEFINITION 2.5 (see [6]). 1. *The tree consisting of a single vertex v is a decomposition tree of the primitive distance-hereditary graph $G = (\{v\}, \emptyset)$.*

2. *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be the decomposition trees of distance-hereditary graphs G_1 and G_2, respectively.*

(a) *If G is formed from G_1 and G_2 by the true twin operation, then a tree $\mathcal{D}$ with the root r represented by $\otimes$ and with the roots of $\mathcal{D}_1$ and $\mathcal{D}_2$ being the two children of r is a decomposition tree of G.*

(b) *If G is formed from G_1 and G_2 by the attachment operation, then a tree $\mathcal{D}$ with the root r represented by $\oplus$ and with the roots of $\mathcal{D}_1$ and $\mathcal{D}_2$ being the left child and the right child of r, respectively, is a decomposition tree of G.*

(c) *If G is formed from G_1 and G_2 by the false twin operation, then a tree $\mathcal{D}$ with the root r represented by $\odot$ and with the roots of $\mathcal{D}_1$ and $\mathcal{D}_2$ being the two children of r is a decomposition tree of G.*

Figure 2.2 shows a distance-hereditary graph and its decomposition tree. Note that the twin set of the given graph is $\{a, b, c, d\}$.

LEMMA 2.6. *A decomposition tree of a distance-hereditary graph can be constructed in $O(n + m)$ sequential time.*

Proof. It follows from the fact that a one-vertex-extension tree can be generated in $O(n + m)$ time [6]. □

3. A general problem-solving paradigm.

3.1. The subgraphs generating problem. Suppose that $G = (V, E)$ is a graph and let $\mathcal{U}$ be the set consisting of all subsets of V. Given a set $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_l\}$, where $Q_i \in \mathcal{U}$, we define MIN_v to be an operator on $\mathcal{Q}$ that returns a set Q_j, for some $1 \leq j \leq l$, such that $|Q_j|$ is the smallest. The operator MAX_v is defined similarly. Given $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_l\}$ and $\mathcal{R} = \{R_1, R_2, \ldots, R_t\}$, where $\mathcal{Q}, \mathcal{R} \subset \mathcal{U}$, $\mathcal{Q}$ and $\mathcal{R}$ are *disjoint* if $Q_i \cap R_j = \emptyset$, $1 \leq i \leq l$, and $1 \leq j \leq t$. For two lists $L_1 = \langle l_1, l_2, \ldots, l_k \rangle$ and $L_1{}' = \langle l_1{}', l_2{}', \ldots, l_j{}' \rangle$, we define the *concatenation of L_1 and $L_1{}'$*, denoted by $L_1 \bullet L_1{}'$, to be the list $\langle l_1, l_2, \ldots, l_k, l_1{}', l_2{}', \ldots, l_j{}' \rangle$.

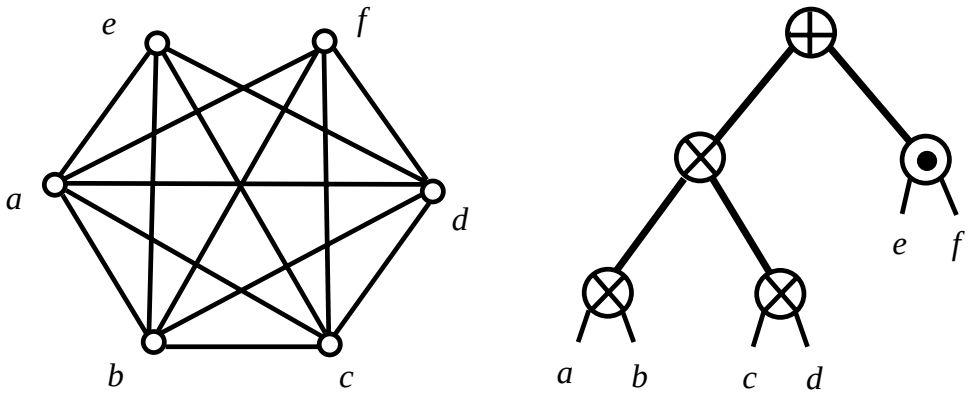

FIG. 2.2. *A distance-hereditary graph with its decomposition tree.*

Consider a rooted tree T. Let $root(T)$ be the root of T. For a node x in T, any node y on the unique path from x to $root(T)$ is called an *ancestor* of x. If y is an ancestor of x, then x is a *descendant* of y. Further, x is a *proper descendant* of y when $x \neq y$. Note that every node is both an ancestor and a descendant of itself. Two nodes in T are *irrelative* if one is not an ancestor of the other. The *least common ancestor* of two nodes x and y in T is the node that is an ancestor to both x and y, and is farthest from $root(T)$.

DEFINITION 3.1. *Let $G = (V, E)$ be a graph and let T be a binary tree. Also let $\mathcal{U}$ be the set consisting of all subsets of V. Given two nonnegative integers k and r, and an operator $\Theta \in \{\text{MIN}_v, \text{MAX}_v\}$, T is an (r, k, Θ)-*subgraph generating tree of G *if the following conditions hold. Let v be a node of T and let N_i be the set of integers from 1 to i.*

1. *Node v is associated with a list of r subgraphs $A_v = \langle A_{v,1}, A_{v,2}, \ldots, A_{v,r}\rangle$ selected from $\mathcal{U}$ such that $|A_{v,i}| = O(1)$ and A_v and A_w are disjoint if v and w are irrelative. These subgraphs are called the* auxiliary subgraphs[1] *of v.*

2. *If v is an internal node, then it is associated with k integers $a_{v,1}, a_{v,2}, \ldots, a_{v,k}$ from N_{r+k}, and the following $2k$ linear unary functions $f_{v,i} : N_{a_{v,i}} \mapsto N_{r+k}$ and $g_{v,i} : N_{a_{v,i}} \mapsto N_{r+k}$, $1 \leq i \leq k$.*

3. *Node v is also associated with a list of k subgraphs $R_v = \langle R_{v,1}, R_{v,2}, \ldots, R_{v,k}\rangle$, called the* target subgraphs *of v, which are defined as follows.*

(a) *If v is a leaf, then R_v is a list of subgraphs selected from $\mathcal{U}$. Moreover, R_x and R_y are disjoint for two arbitrary distinct leaves x and y.*

(b) *If v is an internal node with the children u and w, then*

$$R_{v,i} = \Theta\{Z_{u,f_{u,i}(1)} \cup Z_{w,g_{w,i}(1)}, Z_{u,f_{u,i}(2)} \cup Z_{w,g_{w,i}(2)}, \ldots, Z_{u,f_{u,i}(a_{v,i})} \cup Z_{w,g_{w,i}(a_{v,i})}\}, \tag{3.1}$$

where $1 \leq i \leq k$, $Z_u = R_u \bullet A_u = \langle Z_{u,1}, Z_{u,2}, \ldots, Z_{u,k+r}\rangle$, $Z_w = R_w \bullet A_w = \langle Z_{w,1}, Z_{w,2}, \ldots, Z_{w,k+r}\rangle$. Note that $Z_{u,f_{u,i}(j)} \cap Z_{w,g_{w,i}(j)} = \emptyset$ for $1 \leq j \leq a_{v,i}$.

For a node v in an (r, k, Θ)-subgraph generating tree T, let $T(v)$ be a subtree of T rooted at v and let G_v be the subgraph of G induced by the leaves of G_v. Also let

[1]For ease of implementation, we allow a subgraph of G to be represented by its vertex set if it has no edge.

$\mathcal{U}_v$ be the set consisting of all subsets of $(\cup_{1\le i\le k} R_{x,i}) \cup (\cup_{1\le i\le r} A_{y,i})$, where x is a leaf of $T(v)$ and $y \in V(T(v))$. Note that $\mathcal{U}_v$ and $\mathcal{U}_w$ are disjoint if v and w are irrelative.

Let G be a distance-hereditary graph. As we will show in sections 4.1–4.5, solving some subgraph optimization problem $\mathcal{P}$ on G can be transformed easily into solving that on a corresponding (r, k, Θ)-subgraph generating tree T of G. According to the essential property of $\mathcal{P}$, each node $v \in V(T)$ can be associated with $r(\ge 0)$ subgraphs A_v of G_v in advance such that the following condition holds. For an internal node v with two children u and w, a solution of $\mathcal{P}$ on G_v can be obtained by selecting a subgraph with the maximum or minimum cardinality (depending on $\mathcal{P}$) from the $2(k+r)$ subgraphs in some combinations of R_u, R_w, A_u, and A_w shown as (3.1). Note that R_v is generated in a bottom-up fashion, and the selection can be implemented according Θ together with $f_{u,i}, f_{w,i}, g_{u,i}$ and $g_{w,i}$, $1 \le i \le k$.

DEFINITION 3.2. *Let T be an (r, k, Θ)-subgraph generating tree.* The (r, k, Θ)-subgraph generating problem *is the problem of finding the k target subgraphs of the root of T.*

LEMMA 3.3. *The (r, k, Θ)-subgraph generating problem can be solved in $O((rk + k^2)n)$ time, where n is the number of vertices of the given tree.*

Proof. Clearly, the problem can be solved by a bottom-up evaluation of the given tree. We now show the complexity. Note that there are $l \le r + k$ subgraphs in (3.1) to generate $R_{v,i}$, $1 \le i \le k$, using Θ. Without loss of generality, assume that v is an internal node. According to (3.1), each term is obtained by the union of two disjoint sets selected using the functions $f_{u,i}$ and $g_{w,i}$, where u and w are the two children of v. Since both $f_{u,i}$ and $g_{w,i}$ are linear unary functions which can be evaluated in $O(1)$ time, the desired l subgraphs can be obtained in $O(r + k)$ time. Next, we explain how to implement Θ to select a target subgraph. We can record the cardinality of each of l subgraphs such that generating $R_{v,i}$ is equivalent to finding the maximum (or minimum) among l values. This can be implemented to run in $O(r+k)$ time. Therefore, generating $R_{v,i}$'s for all $1 \le i \le k$ takes $O(rk + k^2)$ time. Since there are totally n vertices in the tree, the problem can be solved with the desired complexity. □

3.2. Parallel complexities of the (r, k, Θ)-subgraph generating problem. In this section, we apply the binary tree contraction technique described in [1] to parallelize the (r, k, Θ)-subgraph generating problem. This technique recursively applies two operations, *prune* and *bypass*, to a given binary tree. $Prune(u)$ is an operation which removes a leaf node u from the current tree, and $bypass(v)$ is an operation (following a prune operation) that removes a node v with exactly one child w and then lets the parent of v become the new parent of w. We define a *contraction phase* to be the consecutive execution of a prune and then bypass operations. Figure 3.1 shows two procedures, $prune(u)$ and $bypass(v)$.

Let T be an n-leave binary tree. Given an Euler tour starting from $root(T)$ of T, the algorithm initially numbers the leaves from 1 to n according to the order of their appearances in the tour. Then the algorithm repeats the following steps. In each step, *prune* and *bypass* work only on the leaves with odd indices and their parents. Hence, these two operations can be performed independently and delete $\lfloor \frac{l}{2} \rfloor$ leaves together with their parents on the binary tree in each step, where l is the number of the current leaves. Therefore, the tree will be reduced to a three-node tree after repeating the steps in $\lceil \log n \rceil$ times.

LEMMA 3.4 (see [1]). *If the prune operation and bypass operation can be performed by one processor in constant time, the binary tree contraction algorithm can*

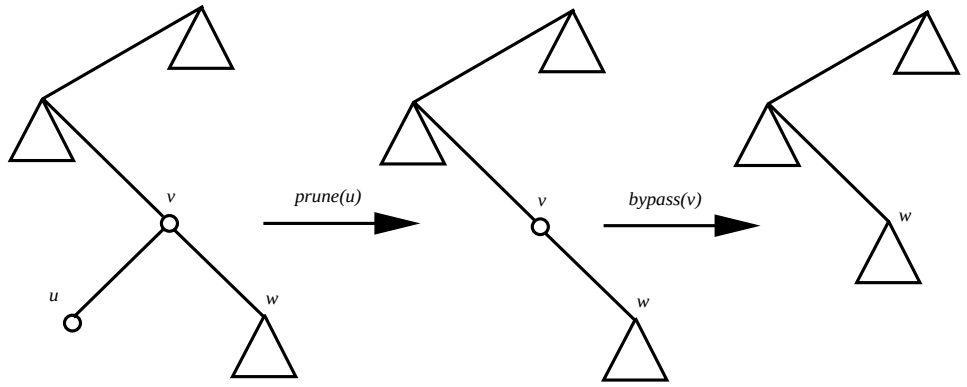

FIG. 3.1. *Illustrating two procedures, prune(u) and bypass(v).*

be implemented in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM, where n is the number of nodes in an input binary tree.

DEFINITION 3.5. *Let u and v be two nodes of an (r,k,Θ)-subgraph generating tree T such that u is a descendant of v. A k-ary function $h: {\mathcal{U}_u}^k \mapsto \mathcal{U}_v$ possesses the* closed form *if $h(X_1,\ldots,X_k)=\Theta\{X_{b_1}\cup C_1, X_{b_2}\cup C_2,\ldots,X_{b_a}\cup C_a, Q\}$, where $b_i\neq b_j$ for two distinct $1\le i,j\le a$, and $C_i, Q\in(\mathcal{U}_v\setminus\mathcal{U}_u)$.*

LEMMA 3.6. *Let $\Theta\in\{\text{MIN}_v,\text{MAX}_v\}$, and let $h_0: {\mathcal{U}_u}^k\mapsto\mathcal{U}_v$ be a function with the closed form, where u is a descendant of v. Let w be a descendant of u and let $h_i: {\mathcal{U}_w}^k\mapsto\mathcal{U}_u$, $1\le i\le k$, be k functions possessing the closed form. Then the function obtained from the composition $h_0\circ(h_1,h_2,\ldots,h_k): {\mathcal{U}_w}^k\mapsto\mathcal{U}_v$ also possesses the closed form.*

Proof. Let $h_i(X_1,\ldots,X_k)=\Theta\{X_{b_1^i}\cup C_1^i, X_{b_2^i}\cup C_2^i,\ldots,X_{b_{a_i}^i}\cup C_{a_i}^i, Q_i\}$ for all $0\le i\le k$. Note that $C_j^i, Q_i\in(\mathcal{U}_u\setminus\mathcal{U}_w)$, $1\le j\le a_i$. Define function $B(i,j)=b_j^i$, where $0\le i\le k$ and $1\le j\le a_i$. We show in the following that $h_0\circ(h_1,\ldots,h_k)$ is a function with the desired form. Clearly,

$$h_0\circ(h_1,\ldots,h_k)=\Theta\{h_{B(0,1)}\cup C_1^0,\ldots,h_{B(0,a_0)}\cup C_{a_0}^0, Q_0\}. \tag{3.2}$$

For $1\le i\le a_0$, $(h_{B(0,i)}\cup C_i^0)(X_1,\ldots,X_k)=\Theta\{X_{B(B(0,i),1)}\cup C_1^{B(0,i)}, X_{B(B(0,i),2)}\cup C_2^{B(0,i)},\ldots,X_{B(B(0,i),a_{B(0,i)})}\cup C_{a_{B(0,i)}}^{B(0,i)}, Q_{B(0,i)}\}\cup C_i^0 = \Theta\{X_{B(B(0,i),1)}\cup(C_1^{B(0,i)}\cup C_i^0),\ldots,X_{B(B(0,i),j)}\cup(C_j^{B(0,i)}\cup C_i^0),\ldots,X_{B(B(0,i),a_{B(0,i)})}\cup(C_{a_{B(0,i)}}^{B(0,i)}\cup C_i^0),(Q_{B(0,i)}\cup C_i^0)\}=\Theta\{X_{B(i',1)}\cup {C'}_1^{i'},\ldots,X_{B(i',a_{i'})}\cup {C'}_{a_{i'}}^{i'}, {Q'}_{i'}\}=h'_{i'}(X_1,\ldots,X_k)$, where $i'=B(0,i)$. We define $\{l_1,l_2,\ldots,l_t\}$, $t\le k$, to be the set of integers such that for each l_s, $1\le s\le t$, there is a term $X_{B(q,p)}$ in $h'_{i'}(X_1,\ldots,X_k)$ with $X_{B(q,p)}=X_{l_s}$ for some p,q. For $1\le s\le t$, let $K_{l_s}=\{X_{B(q,p)}\cup {C'}_p^q \mid X_{B(q,p)}=X_{l_s}\}$ and let ${K'}_{l_s}=\{{C'}_p^q \mid (X_{B(q,p)}\cup {C'}_p^q)\in K_{l_s}\}$. Since ${C'}_p^q\in(\mathcal{U}_v\setminus\mathcal{U}_w)$, each set in ${K'}_{l_s}$ is disjoint with $\mathcal{U}_w$. Notice that X_{l_s} is drawn from $\mathcal{U}_w$. Therefore, (3.2) can be further simplified as follows: $\Theta\{X_{l_1}\cup\Theta\{{K'}_{l_1}\},\ldots,X_{l_t}\cup\Theta\{{K'}_{l_t}\},\Theta\{Q_0,{Q'}_1,{Q'}_2,\ldots,{Q'}_{a_0}\}\}=\Theta\{X_{l_1}\cup D_{l_1},\ldots,X_{l_t}\cup D_{l_t},R\}$, where $D_{l_i}=\Theta\{{K'}_{l_i}\}$, for $1\le i\le t$ and $R=\Theta\{Q_0,{Q'}_1,{Q'}_2,\ldots,{Q'}_{a_0}\}$. It is easy to check that $D_{l_i},R\in\mathcal{U}_v\setminus\mathcal{U}_w$, and w is a descendant of v. Hence, $h_0\circ(h_1,\ldots,h_k)$ possesses the desired form. ∎

We next develop a parallel algorithm for the (r,k,Θ)-subgraph generating problem. For a node x in the current tree H, let $par_H(x)$ ($child_H(x)$) denote the *parent*

(*children*) of x and let $sib_H(x)$ denote the *sibling* of x. The subscript H can be omitted if no ambiguity arises. Also let $H(x)$ denote the subtree of H rooted at x. Recall that $R_x = \langle R_{x,1}, \ldots, R_{x,k}\rangle$ (respectively, $A_x = \langle A_{x,1}, \ldots, A_{x,r}\rangle$) is the list of the target (respectively, auxiliary) subgraphs associated with x.

During the process of executing the tree contraction, we aim at constructing k k-ary functions $h_{x,1}, h_{x,2}, \ldots, h_{x,k}$ associated with each node x of the current tree such that each $h_{x,i}$, $1 \le i \le k$, possesses the closed form and satisfies the condition described below. Let v be an internal node in the current tree whose left child and right child are u and w, respectively. Let u' be the left child and w' be the right child of v in the original tree. Note that u' and w' are ancestors of u and w in the original tree, respectively. For the remainder of this section, we call u' and w' *replacing ancestors of u and w with respect to v*, respectively. Once $R_{u,i}$ and $R_{w,i}$, $1 \le i \le k$, are provided as the inputs of $h_{u,i}$ and $h_{w,i}$, respectively, the target subgraphs of v can be obtained from $Z_{u'} = \langle h_{u,1}(R_{u,1}, \ldots, R_{u,k}), \ldots, h_{u,k}(R_{u,1}, \ldots, R_{u,k})\rangle \bullet A_{u'}$, and $Z_{w'} = \langle h_{w,1}(R_{w,1}, \ldots, R_{w,k}), \ldots, h_{w,k}(R_{w,1}, \ldots, R_{w,k})\rangle \bullet A_{w'}$, using the formula

$$R_{v,i} = \Theta\{Z_{u',f_{u',i}(1)} \cup Z_{w',g_{w',i}(1)}, Z_{u',f_{u',i}(2)} \cup Z_{w',g_{w',i}(2)}, \ldots, Z_{u',f_{u',i}(a_{v,i})} \cup Z_{w',g_{w',i}(a_{v,i})}\}. \tag{3.3}$$

That is, $R_{u'} = \langle R_{u',1}, \ldots, R_{u',k}\rangle = \langle h_{u,1}(R_{u,1}, \ldots, R_{u,k}), \ldots, h_{u,k}(R_{u,1}, \ldots, R_{u,k})\rangle$ and $R_{w'} = \langle R_{w',1}, \ldots, R_{w',k}\rangle = \langle h_{w,1}(R_{w,1}, \ldots, R_{w,k}), \ldots, h_{w,k}(R_{w,1}, \ldots, R_{w,k})\rangle$. We call the above functions $h_{x,i}$, $1 \le i \le k$, computed for each node x in the current tree *the crucial functions of x*. For ease of describing the concept of the crucial function, we demonstrate an example as follows.

Example 1. Consider an internal node v in the original tree T whose left child and right child are u' and w', respectively. Let u be a proper descendant of u' which is a leaf and let w be a proper descendant of w' (see also Figure 3.2(a)). Initially, the k target subgraphs R_v can be obtained by merging $\langle h_{u',1}(R_{u',1}, \ldots, R_{u',k}), \ldots, h_{u',k}(R_{u',1}, \ldots, R_{u',k})\rangle \bullet A_{u'}$ and $\langle h_{w',1}(R_{w',1}, \ldots, R_{w',k}), \ldots, h_{w',k}(R_{w',1}, \ldots, R_{w',k})\rangle \bullet A_{w'}$ in which $R_{u'} = \langle R_{u',1}, \ldots, R_{u',k}\rangle$ and $R_{w'} = \langle R_{w',1}, \ldots, R_{w',k}\rangle$ are indeterminate. After a sequence of contraction phases, assume T' is the current tree in which the left child and the right child of v are u and w, respectively (see also Figure 3.2(b)). Notice that u' and w' are now replacing ancestors of u and w with respect to v, respectively. R_v are then obtained by merging $\langle h_{u,1}(R_{u,1}, \ldots, R_{u,k}), \ldots, h_{u,k}(R_{u,1}, \ldots, R_{u,k})\rangle \bullet A_{u'}$ and $\langle h_{w,1}(R_{w,1}, \ldots, R_{w,k}), \ldots, h_{w,k}(R_{w,1}, \ldots, R_{w,k})\rangle \bullet A_{w'}$. Since $R_{u,i}$ are those subgraphs associated with T before executing the tree contraction, the indeterminate part for generating R_v is reduced to $R_w = \langle R_{w,1}, \ldots, R_{w,k}\rangle$. This part is smaller than the previous one.

We next describe the details of our algorithm. Initially, for each node v in the given tree we construct k functions $h_{v,i}(X_1, \ldots, X_k) = \Theta\{X_i \cup \emptyset, \emptyset\}$, $1 \le i \le k$. Clearly, these functions are crucial functions.

In the execution of the tree contraction, assume that $prune(u)$ and $bypass(par(u))$ are performed consecutively. Let $par(u) = v$ and $sib(u) = w$ in the current tree. Let u' and w' be the replacing ancestors of u and w with respect to v, respectively. Assume that $h_{u,i}$ and $h_{w,i}$, $1 \le i \le k$, are crucial functions of u and w in the current tree. Thus $R_{u'} = \langle h_{u,1}(R_{u,1}, \ldots, R_{u,k}), \ldots, h_{u,k}(R_{u,1}, \ldots, R_{u,k})\rangle$ and $R_{w'} = \langle h_{w,1}(R_{w,1}, \ldots, R_{w,k}), \ldots, h_{w,k}(R_{w,1}, \ldots, R_{w,k})\rangle$. Since u is a leaf, $R_{u,i}$'s are associated with u before executing the tree contraction algorithm. Therefore, the above k target subgraphs $R_{u'}$ can be obtained through function evaluation. On the other hand, since w is not a leaf in the current tree, $R_{w,i}$, $1 \le i \le k$, is an

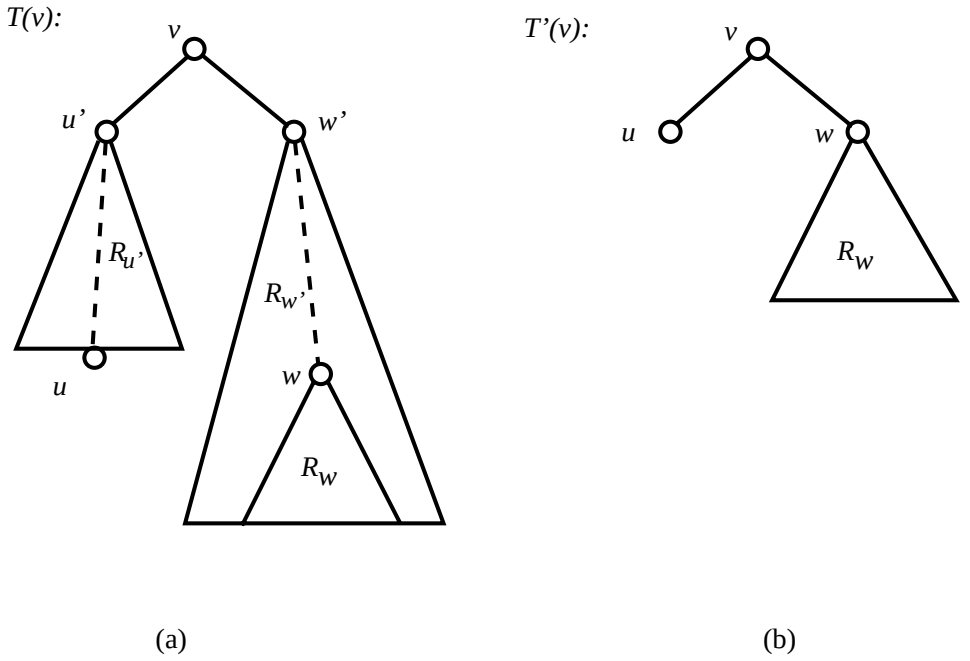

FIG. 3.2. *The concept of crucial functions. The indeterminate part for evaluating R_v shown in* (a) *is smaller than that shown in* (b).

indeterminate value represented by variable X_i. Hence, $R_{w'}$ can be represented by $\langle h_{w,1}(X_1,\ldots,X_k),\ldots,h_{w,k}(X_1,\ldots,X_k)\rangle$. By (3.1), we construct k intermediate functions representing the k target subgraphs of v from those of u' and w' by

$$R_{v,i} = \Theta\{Z_{u',f_{u',i}(1)} \cup Z_{w',g_{w',i}(1)}, Z_{u',f_{u',i}(2)} \cup Z_{w',g_{w',i}(2)},\ldots, Z_{u',f_{u',i}(a_{v,i})} \cup Z_{w',g_{w',i}(a_{v,i})}\}, \tag{3.4}$$

where $Z_{u',f_{u',i}(j)} = R_{u',f_{u',i}(j)} = h_{u,f_{u',i}(j)}(R_{u,1},\ldots,R_{u,k})$, $Z_{w',g_{w',i}(j)} \in A_{w'}$ if $g_{w',i}(j) > k$, and $Z_{w',g_{w',i}(j)} = h_{w,g_{w',i}(j)}(X_1,\ldots,X_k)$ if $g_{w',i}(j) \le k$.

Similar to the proof of Lemma 3.6, (3.4) can be further simplified as

$$R_{v,i} = \Theta\{X_{b_1} \cup C_1, X_{b_2} \cup C_2,\ldots,X_{b_a} \cup C_a, Q\}, \tag{3.5}$$

where $b_i \neq b_j$ for two distinct $1 \le i,\ j \le a$, X_{b_i} are variables drawn from $\mathcal{U}_w$, and $C_i, Q \in (\mathcal{U}_v \setminus \mathcal{U}_w)$.

Therefore, the above functions (constructed after executing $prune(u)$) possess the closed form. Given those functions $R_{v,i}$'s, the contribution to the k target subgraphs of $par(v)$ is obtained by function composition $h_{v,i}(R_{v,1},\ldots,R_{v,k})$ for all $1 \le i \le k$. These functions are constructed for w after executing $bypass(par(v))$. By Lemma 3.6, $h_{v,i}(R_{v,1},\ldots,R_{v,k})$, $1 \le i \le k$, possesses the closed form. Hence, we have the following lemma.

LEMMA 3.7. *During the process of executing the binary tree contraction on an (r,k,Θ)-subgraph generating tree to remove some nodes, the crucial functions of the remaining nodes of the current tree can be constructed in $O(k^2(r+k))$ time using one processor.*

Proof. This can be shown by induction on the number of contraction phases based on the arguments preceding the lemma. For constructing each of the k functions, there are at most $k(r+k)$ terms generated. These terms can be simplified as the closed form using Θ. Thus the desired complexities follow. □

THEOREM 3.8. *The (r,k,Θ)-subgraph generating problem can be solved in $O(k^2(r+k)\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM, where n is the number of nodes of the input tree.*

Proof. The algorithm for solving the (r,k,Θ)-subgraph generating problem consists of an initial assignment of k crucial functions to each node of the input tree, and an application of the tree contraction algorithm such that the crucial functions after executing $prune(v)$ and $bypass(par(v))$ are constructed by Lemma 3.7. Once the algorithm terminates, a three-node tree T' results. Let t be the root of T' and y, z be two children of t. Note that the k target subgraphs of y' and z', the replacing ancestors of y and z with respect to t, can be generated by their corresponding crucial functions. Moreover, the auxiliary subgraphs associated with y' and z' before executing the tree contraction are now maintained in y and z by (3.3). Therefore, according to the operators associated with t, the k target subgraphs of t can be generated. By Lemmas 3.4 and 3.7, the problem can be solved with the stated complexities. □

DEFINITION 3.9. *Let G be a distance-hereditary graph. A problem $\mathcal{P}$ is said to be an* (r,k,Θ)-regular problem *on G if $\mathcal{P}$ can be reduced to an (r,k,Θ)-subgraph generating problem $\mathcal{B}$ on a decomposition tree of G such that the following two conditions hold.*

1. *The solution of $\mathcal{B}$ is exactly the solution of $\mathcal{P}$.*
2. *The reduction scheme takes $O(k^2(r+k)\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM, where n is the number of nodes in the given decomposition tree.*

Note that each (r,k,Θ)-regular problem corresponds to an (r,k,Θ)-subgraph generating tree. This tree is obtained from a decomposition tree $\mathcal{D}_G$ in which some additional data structures are associated with $V(\mathcal{D}_G)$ (refer to Definition 3.1). In the remainder of this section and section 4, we assume that a decomposition tree is given for solving an (r,k,Θ)-regular problem on a distance-hereditary graph. Such a tree will be constructed using a parallel algorithm presented in section 5.

LEMMA 3.10. *Given a decomposition tree of a distance-hereditary graph G, an (r,k,Θ)-regular problem on G can be solved in $O(k^2(r+k)\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM.*

Proof. The proof follows from Definition 3.9 and Theorem 3.8. □

LEMMA 3.11. *An (r,k,Θ)-regular problem on a distance-hereditary graph can be solved in $O(k(r+k)n+m)$ sequential time.*

Proof. According to Lemma 2.6, a corresponding (r,k,Θ)-subgraph generating tree can be constructed in $O(k(r+k)n+m)$ time. By Lemma 3.3, an (r,k,Θ)-regular problem can be solved within the desired complexity. □

4. (r,k,Θ)-regular problems. Given a problem $\mathcal{P}$, a graph G, a subgraph H of G, and a subset S of vertices in H, $\mathcal{P}_S(G,H)$ is a solution to the problem such that this solution has a nonempty intersection with S and is contained in H. For the case of $S=\emptyset$, i.e., $\mathcal{P}_\emptyset(G,H)$, the notation represents a solution to G, and this solution is contained in H. For brevity, let $\mathcal{P}_S(G,G)=\mathcal{P}_S(G)$.

In this section, let $G=(V,E)$ be a distance-hereditary graph and S be the twin set of G. Also let $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ be distance-hereditary graphs with the twin sets S_1 and S_2, respectively. Recall that $S=S_1$ if $G=G_1\oplus G_2$. We will show that the problems demonstrated in sections 4.1–4.5 can be efficiently

parallelized using our strategy. We also note that combining the results of [9, 14], the sequential linear time complexity of these problems can also be obtained.

4.1. The maximum clique problem. A graph is a *clique* if there is an edge between every pair of vertices. We say a clique is in G if it is an induced subgraph of G. We define the *maximum clique problem* $\mathcal{C}$ to be the problem that finds a clique with the maximum number of vertices in the input graph. A previous work to solve this problem on distance-hereditary graph can be found in [19]. Using our notation, we want to solve $\mathcal{C}_\emptyset(G)$. For a primitive distance-hereditary graph $G = (\{v\}, \emptyset)$, $\mathcal{C}_\emptyset(G) = \mathcal{C}_\emptyset(G, G[S]) = \{v\}$.

THEOREM 4.1.

1. *In the case of* $G = G_1 \otimes G_2$,
   - $\mathcal{C}_\emptyset(G) = \text{MAX}_v\{\mathcal{C}_\emptyset(G_1), \mathcal{C}_\emptyset(G_2), \mathcal{C}_\emptyset(G_1, G_1[S_1]) \cup \mathcal{C}_\emptyset(G_2, G_2[S_2])\}$;
   - $\mathcal{C}_\emptyset(G, G[S]) = \mathcal{C}_\emptyset(G_1, G_1[S_1]) \cup \mathcal{C}_\emptyset(G_2, G_2[S_2])$.
2. *In the case of* $G = G_1 \oplus G_2$,
   - $\mathcal{C}_\emptyset(G) = \text{MAX}_v\{\mathcal{C}_\emptyset(G_1), \mathcal{C}_\emptyset(G_2), \mathcal{C}_\emptyset(G_1, G_1[S_1]) \cup \mathcal{C}_\emptyset(G_2, G_2[S_2])\}$;
   - $\mathcal{C}_\emptyset(G, G[S]) = \mathcal{C}_\emptyset(G_1, G_1[S_1])$.
3. *In the case of* $G = G_1 \odot G_2$,
   - $\mathcal{C}_\emptyset(G) = \text{MAX}_v\{\mathcal{C}_\emptyset(G_1), \mathcal{C}_\emptyset(G_2)\}$;
   - $\mathcal{C}_\emptyset(G, G[S]) = \text{MAX}_v\{\mathcal{C}_\emptyset(G_1, G_1[S_1]), \mathcal{C}_\emptyset(G_2, G_2[S_2])\}$.

Proof. The proof is straightforward. □

For a node v in a decomposition tree $\mathcal{D}_G$, recall that G_v denote a subgraph of G induced by the leaves of the subtree of $\mathcal{D}_G$ rooted at v. Let S_v denote the twin set of G_v. For convenience, let $V(G_v) = V_v$.

THEOREM 4.2. *The maximum clique problem is a* $(1, 2, \text{MAX}_v)$*-regular problem on distance-hereditary graphs.*

Proof. We first reduce the problem to a $(1, 2, \text{MAX}_v)$-subgraph generating problem. A corresponding $(1, 2, \text{MAX}_v)$-subgraph generating tree can be constructed by the following steps:

(S1) For each node $v \in V(\mathcal{D}_G)$, set $A_v = \langle\emptyset\rangle$.

(S2) For each internal node v, let u and w be the left child and the right child of v. For $x \in \{u, w\}$, let $Z_x = R_x \bullet A_x = \langle Z_{x,1}, Z_{x,2}, Z_{x,3}\rangle = \langle\mathcal{C}_\emptyset(G_x), \mathcal{C}_\emptyset(G_x, G_x[S_x]), \emptyset\rangle$. Set two integers $a_{v,1}, a_{v,2}$ and construct functions $f_{x,i}$ and $g_{x,i}$, $1 \le i \le 2$, according to the following cases:

CASE 1: v is a $\otimes$-node. Set $a_{v,1} = 3, a_{v,2} = 1$, and $f_{u,1}(1) = g_{w,1}(2) = 1, f_{u,1}(3) = g_{w,1}(3) = f_{u,2}(1) = g_{w,2}(1) = 2, f_{u,1}(2) = g_{w,1}(1) = 3$.
According to Theorem 4.1(1), $\mathcal{C}_\emptyset(G_v) = R_{v,1} = \text{MAX}_v\{Z_{u,f_{u,1}(1)} \cup Z_{w,g_{w,1}(1)}, Z_{u,f_{u,1}(2)} \cup Z_{w,g_{w,1}(2)}, \ldots, Z_{u,f_{u,1}(a_{v,1})} \cup Z_{w,g_{w,1}(a_{v,1})}\} = \text{MAX}_v\{Z_{u,1} \cup Z_{w,3}, Z_{u,3} \cup Z_{w,1}, Z_{u,2} \cup Z_{w,2}\}$, and $\mathcal{C}_\emptyset(G_v, G_v[S_v]) = R_{v,2} = \text{MAX}_v\{Z_{u,f_{u,2}(1)} \cup Z_{w,g_{w,2}(1)}, Z_{u,f_{u,2}(2)} \cup Z_{w,g_{w,2}(2)}, \ldots, Z_{u,f_{u,2}(a_{v,2})} \cup Z_{w,g_{w,2}(a_{v,2})}\} = \text{MAX}_v\{Z_{u,2} \cup Z_{w,2}\}$.

CASE 2: v is a $\oplus$-node. Set $a_{v,1} = 3, a_{v,2} = 1$, and $f_{u,1}(1) = g_{w,1}(2) = 1, f_{u,1}(3) = g_{w,1}(3) = f_{u,2}(1) = 2, f_{u,1}(2) = g_{w,1}(1) = g_{w,2}(1) = 3$.
Then, $\mathcal{C}_\emptyset(G_v) = R_{v,1} = \text{MAX}_v\{Z_{u,1} \cup Z_{w,3}, Z_{u,3} \cup Z_{w,1}, Z_{u,2} \cup Z_{w,2}\}$ and $\mathcal{C}_\emptyset(G_v, G_v[S_v]) = R_{v,2} = \text{MAX}_v\{Z_{u,2} \cup Z_{w,3}\}$.

CASE 3: v is a $\odot$-node. Set $a_{v,1} = 2, a_{v,2} = 2$, and $f_{u,1}(1) = g_{w,1}(2) = 1, f_{u,2}(1) = g_{w,2}(2) = 2, f_{u,1}(2) = g_{w,1}(1) = g_{w,2}(1) = f_{u,2}(2) = 3$.
Then, $\mathcal{C}_\emptyset(G_v) = R_{v,1} = \text{MAX}_v\{Z_{u,1} \cup Z_{w,3}, Z_{u,3} \cup Z_{w,1}\}$ and $\mathcal{C}_\emptyset(G_v, G_v[S_v]) = R_{v,2} = \text{MAX}_v\{Z_{u,2} \cup Z_{w,3}, Z_{u,3} \cup Z_{w,2}\}$.

(S3) For each leaf l corresponding to a primitive distance-hereditary graph $G_l = (\{v\}, \emptyset)$, set two target subgraphs of l to be $R_l = \langle R_{l,1}, R_{l,2}\rangle = \langle\mathcal{C}_\emptyset(G_l), \mathcal{C}_\emptyset(G_l, G_l[S_l])\rangle$

$= \langle\{v\}, \{v\}\rangle$.
The other two cases for $\oplus$-node and $\odot$-node can be verified similarly. Therefore, the maximum clique problem can be reduced to a $(1, 2, \text{MAX}_v)$-subgraph generating problem. Clearly, steps (S1)–(S3) can be implemented in $O(1)$ time using $O(n)$ processors on an EREW PRAM. As with the aid of Brent's scheduling principle [22], the reduction scheme takes $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. By Definition 3.9, the theorem holds. □

4.2. The maximum independent set problem. An *independent set* of a graph is a subset of its vertices such that no two vertices in the subset are adjacent. The *maximum independent set problem* $\mathcal{I}$ is the problem of finding a maximum cardinality independent set in the input graph. A previous sequential linear time algorithm to solve this problem on distance-hereditary graphs can be found in [15]. Using our notation, given an input graph G, a solution is $\mathcal{I}_\emptyset(G)$. For a primitive distance-hereditary graph $G = (\{v\}, \emptyset)$, $\mathcal{I}_\emptyset(G)$ and $\mathcal{I}_S(G)$ are both equal to $\{v\}$, and $\mathcal{I}_\emptyset(G, G[V \setminus S]) = \emptyset$.

THEOREM 4.3.

1. *In the case of* $G = G_1 \otimes G_2$,
   - $\mathcal{I}_\emptyset(G) = \text{MAX}_v\{\mathcal{I}_{S_1}(G_1) \cup \mathcal{I}_\emptyset(G_2, G_2[V_2 \setminus S_2]),$
     $\mathcal{I}_{S_2}(G_2) \cup \mathcal{I}_\emptyset(G_1, G_1[V_1 \setminus S_1]), \mathcal{I}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{I}_\emptyset(G_2, G_2[V_2 \setminus S_2])\}$;
   - $\mathcal{I}_S(G) = \text{MAX}_v\{\mathcal{I}_{S_1}(G_1) \cup \mathcal{I}_\emptyset(G_2, G_2[V_2 \setminus S_2]), \mathcal{I}_{S_2}(G_2) \cup \mathcal{I}_\emptyset(G_1, G_1[V_1 \setminus S_1])\}$;
   - $\mathcal{I}_\emptyset(G, G[V \setminus S]) = \mathcal{I}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{I}_\emptyset(G_2, G_2[V_2 \setminus S_2])$.
2. *In the case of* $G = G_1 \oplus G_2$,
   - $\mathcal{I}_\emptyset(G) = \text{MAX}_v\{\mathcal{I}_{S_1}(G_1) \cup \mathcal{I}_\emptyset(G_2, G_2[V_2 \setminus S_2]),$
     $\mathcal{I}_{S_2}(G_2) \cup \mathcal{I}_\emptyset(G_1, G_1[V_1 \setminus S_1]), \mathcal{I}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{I}_\emptyset(G_2, G_2[V_2 \setminus S_2])\}$;
   - $\mathcal{I}_S(G) = \mathcal{I}_{S_1}(G_1) \cup \mathcal{I}_\emptyset(G_2, G_2[V_2 \setminus V_2])$;
   - $\mathcal{I}_\emptyset(G, G[V \setminus S]) = \mathcal{I}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{I}_\emptyset(G_2)$.
3. *In the case of* $G = G_1 \odot G_2$,
   - $\mathcal{I}_\emptyset(G) = \mathcal{I}_\emptyset(G_1) \cup \mathcal{I}_\emptyset(G_2)$;
   - $\mathcal{I}_S(G) = \text{MAX}_v\{\mathcal{I}_{S_1}(G_1) \cup \mathcal{I}_\emptyset(G_2), \mathcal{I}_{S_2}(G_2) \cup \mathcal{I}_\emptyset(G_1)\}$;
   - $\mathcal{I}_\emptyset(G, G[V \setminus S]) = \mathcal{I}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{I}_\emptyset(G_2, G_2[V_2 \setminus S_2])$.

Proof. The proof is straightforward. □

As with the proof similar to that of Theorem 4.2, the following result can be obtained.

THEOREM 4.4. *The maximum independent set problem is a* $(0, 3, \text{MAX}_v)$*-regular problem on distance-hereditary graphs.*

4.3. The vertex connectivity problem. We now consider the vertex connectivity problem. A *vertex separator* (separator for short) of a graph is a set of vertices whose removal increases the number of connected components or results in a trivial graph, i.e., a graph with no edges. A vertex separator Q of G is *minimal* if any proper subset of Q is not a vertex separator of G. A *minimum vertex separator* of G is a vertex separator with the minimum cardinality. We define the *vertex connectivity problem* $\mathcal{V}$ to be the problem that finds a minimum vertex separator for the input graph. A related work can be found in [26]. Using our notation, a solution on the input graph G is denoted as $\mathcal{V}_\emptyset(G)$.

LEMMA 4.5. *Let* Q *be a minimal vertex separator of* G *such that* $G = G_1 \otimes G_2$ *or* $G = G_1 \oplus G_2$. *If* $S_1 \neq V(G_1)$ *and* $S_2 \neq V(G_2)$, *then there is a connected component* H *of* $G[V \setminus Q]$ *such that* $V(H) \cap (S_1 \cup S_2) = \emptyset$.

Proof. Note that G is connected and both $G[V \setminus S_1]$ and $G[V \setminus S_2]$ are disconnected. Thus $S_1 \not\subset Q$ and $S_2 \not\subset Q$. By assumption, $S_1 \neq V(G_1)$ and $S_2 \neq V(G_2)$, and this lemma holds trivially when $S_1 = Q$ or $S_2 = Q$. We now assume $S_1 \neq Q$ and $S_2 \neq Q$. Hence there is a vertex of S_1 and one of S_2 in $G[V \setminus Q]$. Assume the contrary, that every connected component C of $G[V \setminus Q]$ satisfies $V(C) \cap (S_1 \cup S_2) \neq \emptyset$. Since every vertex of S_1 is connected to all the vertices of S_2, $G[V \setminus Q]$ remains connected which contradicts the fact that Q is a vertex separator of G. □

For a subset X of V, let $N_G(X) = (\bigcup_{v \in X} N_G(v)) \setminus X$. The subscript G in the notations used in this section can be omitted when no ambiguity arises.

LEMMA 4.6. *Let Q be a minimal vertex separator of G. If $S_1 \neq V(G_1)$ and $S_2 \neq V(G_2)$, then $Q \subseteq V(G_i)$ for some $i \in \{1, 2\}$.*

Proof. If $G = G_1 \odot G_2$, the result holds clearly; otherwise, $G = G_1 \otimes G_2$ or $G = G_1 \oplus G_2$. By Lemma 4.5, there exists a connected component H of $G[V \setminus Q]$ such that $V(H) \subseteq V(G_i)$ for some $i \in \{1, 2\}$. Since G is connected and Q is a minimal vertex separator, $N(V(H)) = Q$. By $V(H) \cap (S_1 \cup S_2) = \emptyset$, we know that $N(V(H)) \subseteq V(G_i)$. Therefore, $Q \subseteq V(G_i)$. □

LEMMA 4.7. *If G is disconnected, then for any connected component C of G, $C \cap S \neq \emptyset$.*

Proof. The proof is straightforward. □

The following lemma can be shown by the structure characterization of distance-hereditary graphs.

LEMMA 4.8. *Let $G = G_1 \otimes G_2$ or $G = G_1 \oplus G_2$. If $V(G_i) = S_i$ and G_j is disconnected, where $i, j \in \{1, 2\}$ and $i \neq j$, then S_i is a minimal vertex separator of G.*

Let inf be an infinite set of vertices. Given a graph G, let $conn(G)$ be inf if G is connected and be $\emptyset$ if G is disconnected. For a distance-hereditary graph G with the twin set S, a vertex separator Q is called *crucial* if there exists a component H of $G[V \setminus Q]$ such that $V(H) \cap S = \emptyset$. Define $\mathcal{V}_S^2(G)$ to be the problem that finds a minimum crucial vertex separator of G. Let $\mathcal{V}_S^2(G)$ be inf if there is no vertex separator satisfying the requirements. Recall that every connected component of G has a nonempty intersection with S. We define $\mathcal{V}_S^3(G)$ to be the problem that returns inf if $S = V(G)$, and returns $\text{MIN}_v\{V(C) \cap S |\ C$ if a connected component of G and $(V(C) \setminus S) \neq \emptyset\}$ otherwise. For a primitive distance-hereditary graph $G = (\{v\}, \emptyset)$, $\mathcal{V}_\emptyset(G) = \emptyset$, $\mathcal{V}_S^2(G) = inf$, and $\mathcal{V}_S^3(G) = inf$.

LEMMA 4.9. *Assume that $G = G_1 \otimes G_2$.*

1. *If $S_1 = V(G_1)$ and $S_2 \neq V(G_2)$, then*
   $\mathcal{V}_\emptyset(G) = \text{MIN}_v\{S_1 \cup conn(G_2), S_1 \cup \mathcal{V}_\emptyset(G_2), \mathcal{V}_{S_2}^2(G_2), \mathcal{V}_{S_2}^3(G_2)\}$,
   $\mathcal{V}_S^2(G) = \text{MIN}_v\{\mathcal{V}_{S_2}^2(G_2), \mathcal{V}_{S_2}^3(G_2)\}$, *and*
   $\mathcal{V}_S^3(G) = S_1 \cup S_2$.
2. *If $S_2 = V(G_2)$ and $S_1 \neq V(G_1)$, then*
   $\mathcal{V}_\emptyset(G) = \text{MIN}_v\{S_2 \cup conn(G_1), S_2 \cup \mathcal{V}_\emptyset(G_1), \mathcal{V}_{S_1}^2(G_1), \mathcal{V}_{S_1}^3(G_1)\}$,
   $\mathcal{V}_S^2(G) = \text{MIN}_v\{\mathcal{V}_{S_1}^2(G_1), \mathcal{V}_{S_1}^3(G_1)\}$, *and*
   $\mathcal{V}_S^3(G) = S_1 \cup S_2$.
3. *If $S_1 = V(G_1)$ and $S_2 = V(G_2)$, then*
   $\mathcal{V}_\emptyset(G) = \text{MIN}_v\{S_1 \cup conn(G_2), S_2 \cup conn(G_1), S_1 \cup \mathcal{V}_\emptyset(G_2), S_2 \cup \mathcal{V}_\emptyset(G_1)\}$,
   $\mathcal{V}_S^2(G) = inf$, *and*
   $\mathcal{V}_S^3(G) = inf$.
4. *If $S_1 \neq V(G_1)$ and $S_2 \neq V(G_2)$, then*
   $\mathcal{V}_\emptyset(G) = \text{MIN}_v\{\ \mathcal{V}_{S_1}^2(G_1), \mathcal{V}_{S_2}^2(G_2),\ \mathcal{V}_{S_1}^3(G_1), \mathcal{V}_{S_2}^3(G_2)\}$,

$\mathcal{V}_S^2(G) = \text{MIN}_v\{\mathcal{V}_{S_1}^2(G_1), \mathcal{V}_{S_2}^2(G_2), \mathcal{V}_{S_1}^3(G_1), \mathcal{V}_{S_2}^3(G_2)\}$,
$\mathcal{V}_S^3(G) = S_1 \cup S_2$.

Proof. We first consider the situation where $S_1 = V(G_1)$ and $S_2 \neq V(G_2)$. Let Q be a minimum vertex separator of G. Note that G is connected. There are five cases.
CASE 1: $Q \subset S_1$. It is impossible because $G[V \setminus Q]$ remains to be connected.
CASE 2: $Q = S_1$. In this case, G_2 must be disconnected. Thus $\mathcal{V}_\emptyset(G)$ equals $S_1 \cup conn(G_2)$.
CASE 3: $Q \cap S_1 \neq \emptyset$ and $S_1 \setminus Q \neq \emptyset$. Clearly, $Q \cap V(G_2) \neq \emptyset$. There are two subcases.
CASE 3.1: $S_2 \subset Q$. This contradicts the fact that $|Q|$ is the minimum because S_2 is also a vertex separator of G.
CASE 3.2: $S_2 \setminus Q \neq \emptyset$. Thus the vertices in $(S_1 \cup S_2) \setminus Q$ are in the same connected component, say H, of $G[V \setminus Q]$. Let H' be another connected component of $G[V \setminus Q]$. Since G is connected and $V(H') \cap (S_1 \cup S_2) = \emptyset$, $N(V(H')) = Q \subseteq V(G_2)$. This contradicts $Q \cap S_1 \neq \emptyset$.
CASE 4: $S_1 \subseteq Q$ and $Q \cap V(G_2) \neq \emptyset$. In this case, G_2 is connected; otherwise, S_1 is a vertex separator of G. Moreover, for every connected component C of $G[V \setminus Q]$, $V(C) \cap S_2 \neq \emptyset$ (otherwise, $Q \setminus S_1$ is a vertex separator of G). Let $Q' = Q \cap V(G_2)$. Clearly, Q' is a minimal vertex separator of $G_2 = (V_2, E_2)$. We next show Q' is a minimum vertex separator of G_2. Assume the contrary, that Q'' is a vertex separator of G_2 such that $|Q''| < |Q'|$. There are two situations.
(a) Every connected component of $G_2[V_2 \setminus Q'']$ has a nonempty intersection with S_2. Clearly, $S_1 \cup Q''$ is a vertex separator of G, and a contradiction arises because $|S_1 \cup Q''| < |S_1 \cup Q'| = Q$.
(b) There exists a connected component H of $G_2[V_2 \setminus Q'']$ with $V(H) \cap S_2 = \emptyset$. Then Q'' is a vertex separator of G and $|Q''| < |Q'| < |Q|$ which contradicts the assumption that Q is a minimum separator of G.
By the above discussion, $\mathcal{V}_\emptyset(G)$ equals $S_1 \cup \mathcal{V}_\emptyset(G_2)$.
CASE 5: $Q \cap S_1 = \emptyset$ (i.e., $Q \subseteq V(G_2)$).
CASE 5.1: Q is a vertex separator of G_2. If every connected component of $G[V \setminus Q]$ has a nonempty intersection with S_2, then $G[V \setminus Q]$ remains connected. This contradicts the fact that Q is a vertex separator of G. Hence, there exists a connected component H of $G[V \setminus Q]$ such that $V(H) \cap S_2 = \emptyset$. This implies $\mathcal{V}_\emptyset(G) = \mathcal{V}_{S_2}^2(G_2)$.
CASE 5.2: Q is not a vertex separator of G_2. There exists a connected component H of $G[V \setminus Q]$ such that $V(H) \cap S_1 = \emptyset$ and $V(H) \cap S_2 = \emptyset$. Note that $N(V(H)) \subset V(G_2)$. Moreover, the subgraph induced by $V(H) \cup Q$ is a connected component, say C, of G_2 and $Q = (S_2 \cap V(C))$ by the facts that Q is not a vertex separator of G_2 and $S_2 \cap V(C)$ is a minimal vertex separator of G. This implies that $\mathcal{V}_\emptyset(G) = \mathcal{V}_{S_2}^3(G_2)$.
Combining the above cases, we have $\mathcal{V}_\emptyset(G) = \text{MIN}_v\{S_1 \cup conn(G_2), S_1 \cup \mathcal{V}_\emptyset(G_2), \mathcal{V}_{S_2}^2(G_2), \mathcal{V}_{S_2}^3(G_2)\}$. The equations for computing $\mathcal{V}_S^2(G)$ and $\mathcal{V}_S^3(G)$ can be shown similarly from the structure characterization of G. By Lemmas 4.5–4.8, the other situations can be shown analogously. □

The following lemma can be shown in a way that is similar to the proof of Lemma 4.9.

LEMMA 4.10. *Assume that* $G = G_1 \oplus G_2$.

1. *If* $S_1 = V(G_1)$ *and* $S_2 \neq V(G_2)$, *then*
$\mathcal{V}_\emptyset(G) = \text{MIN}_v\{S_1 \cup conn(G_2), S_1 \cup \mathcal{V}_\emptyset(G_2), \mathcal{V}_{S_2}^2(G_2), \mathcal{V}_{S_2}^3(G_2)\}$,
$\mathcal{V}_S^2(G) = \text{MIN}_v\{S_1 \cup conn(G_2), S_1 \cup \mathcal{V}_\emptyset(G_2), \mathcal{V}_{S_2}^2(G_2), \mathcal{V}_{S_2}^3(G_2)\}$, *and*

$\mathcal{V}^3_S(G) = S_1$.

2. *If* $S_2 = V(G_2)$ *and* $S_1 \neq V(G_1)$, *then*
$\mathcal{V}_\emptyset(G) = \text{MIN}_v\{S_2 \cup conn(G_1), S_2 \cup \mathcal{V}_\emptyset(G_1), \mathcal{V}^2_{S_1}(G_1), \mathcal{V}^3_{S_1}(G_1)\}$,
$\mathcal{V}^2_S(G) = \text{MIN}_v\{\mathcal{V}^2_{S_1}(G_1), \mathcal{V}^3_{S_1}(G_1)\}$, *and*
$\mathcal{V}^3_S(G) = S_1$.
3. *If* $S_1 = V(G_1)$ *and* $S_2 = V(G_2)$, *then*
$\mathcal{V}_\emptyset(G) = \text{MIN}_v\{S_1 \cup conn(G_2), S_2 \cup conn(G_1), S_1 \cup \mathcal{V}_\emptyset(G_2), S_2 \cup \mathcal{V}_\emptyset(G_1)\}$,
$\mathcal{V}^2_S(G) = \text{MIN}_v\{S_1 \cup conn(G_2), S_1 \cup \mathcal{V}_\emptyset(G_2)\}$, *and*
$\mathcal{V}^3_S(G) = S_1$.
4. *If* $S_1 \neq V(G_1)$ *and* $S_2 \neq V(G_2)$, *then*
$\mathcal{V}_\emptyset(G) = \text{MIN}_v\{\mathcal{V}^2_{S_1}(G_1), \mathcal{V}^2_{S_2}(G_2), \mathcal{V}^3_{S_1}(G_1), \mathcal{V}^3_{S_2}(G_2)\}$,
$\mathcal{V}^2_S(G) = \text{MIN}_v\{\mathcal{V}^2_{S_1}(G_1), \mathcal{V}^2_{S_2}(G_2), \mathcal{V}^3_{S_1}(G_1), \mathcal{V}^3_{S_2}(G_2)\}$, *and*
$\mathcal{V}^3_S(G) = S_1$.

LEMMA 4.11. *Assume that* $G = G_1 \odot G_2$.

1. *If* $S_1 = V(G_1)$ *and* $S_2 \neq V(G_2)$, *then*
$\mathcal{V}_\emptyset(G) = \text{MIN}_v\{\mathcal{V}_\emptyset(G_1), \mathcal{V}_\emptyset(G_2)\}$,
$\mathcal{V}^2_S(G) = \mathcal{V}^2_{S_2}(G_2)$, *and*
$\mathcal{V}^3_S(G) = \mathcal{V}^3_{S_2}(G_2)$.
2. *If* $S_2 = V(G_2)$ *and* $S_1 \neq V(G_1)$, *then*
$\mathcal{V}_\emptyset(G) = \text{MIN}_v\{\mathcal{V}_\emptyset(G_1), \mathcal{V}_\emptyset(G_2)\}$,
$\mathcal{V}^2_S(G) = \mathcal{V}^2_{S_1}(G_1)$, *and*
$\mathcal{V}^3_S(G) = \mathcal{V}^3_{S_1}(G_1)$.
3. *If* $S_1 = V(G_1)$ *and* $S_2 = V(G_2)$, *then*
$\mathcal{V}_\emptyset(G) = \text{MIN}_v\{\mathcal{V}_\emptyset(G_1), \mathcal{V}_\emptyset(G_2)\}$,
$\mathcal{V}^2_S(G) = inf$, *and*
$\mathcal{V}^3_S(G) = inf$.
4. *If* $S_1 \neq V(G_1)$ *and* $S_2 \neq V(G_2)$, *then*
$\mathcal{V}_\emptyset(G) = \text{MIN}_v\{\mathcal{V}_\emptyset(G_1), \mathcal{V}_\emptyset(G_2)\}$,
$\mathcal{V}^2_S(G) = \text{MIN}_v\{\mathcal{V}^2_{S_1}(G_1), \mathcal{V}^2_{S_2}(G_2)\}$, *and*
$\mathcal{V}^3_S(G) = \text{MIN}_v\{\mathcal{V}^3_{S_1}(G_1), \mathcal{V}^3_{S_2}(G_2)\}$.

Proof. The proof follows from the definition of the vertex separator and Lemma 4.6. □

THEOREM 4.12. *The vertex connectivity problem is a* $(2, 4, \text{MIN}_v)$*-regular problem on distance-hereditary graphs.*

Proof. We first reduce the problem to a $(2, 4, \text{MIN}_v)$-regular problem. A corresponding $(2, 4, \text{MIN}_v)$-subgraph generating tree can be constructed by the following steps:

(S1) For each node $v \in V(\mathcal{D}_G)$, determine whether $S_v = V(G_v)$ and determine whether G_v is connected.

(S2) For each node $v \in V(\mathcal{D}_G)$, set $A_v = \langle \emptyset, inf \rangle$.[2]

(S3) For each internal node v, let u and w be the left child and the right child of v, respectively. Set four integers $a_{v,1}, \ldots, a_{v,4}$ and functions $f_{x,i}$ and $g_{x,i}$, where $x \in \{u, w\}$ and $1 \leq i \leq 4$, according to Lemmas 4.9–4.11. Without loss of generality, assume that v is a $\otimes$-node. (The case of v being a $\oplus$- or $\odot$-node can be shown similarly.) There are four cases corresponding to 1–4 of Lemma 4.9. Here we consider only that $S_1 = V(G_1)$ and $S_2 \neq V(G_2)$. The other cases are analogous. Let $\mathcal{V}_\emptyset(G_v) = R_{v,1}, \mathcal{V}^2_{S_v}(G_v) = R_{v,2}, \mathcal{V}^3_{S_v}(G_v) = R_{v,3}$, and $S_v = R_{v,4}$, and let

[2] It is not difficult to generalize the (r, k, Θ)-subgraph generating tree problem when the input is inf.

$Z_v = R_v \bullet A_v = \langle Z_{v,1}, \ldots, Z_{v,6}\rangle = \langle \mathcal{V}_\emptyset(G_v), \mathcal{V}^2_{S_v}(G_v), \mathcal{V}^3_{S_v}(G_v), S_v, \emptyset, inf\rangle$. Consider the following two cases.

CASE 1: G_w is connected. In this case, $\mathcal{V}_\emptyset(G) = \text{MIN}_v\{S_1 \cup conn(G_2), S_1 \cup \mathcal{V}_\emptyset(G_2), \mathcal{V}^2_{S_2}(G_2), \mathcal{V}^3_{S_2}(G_2)\} = \text{MIN}_v\{S_1 \cup \mathcal{V}_\emptyset(G_2), \mathcal{V}^2_{S_2}(G_2), \mathcal{V}^3_{S_2}(G_2)\}$ because $conn(G_2) = inf$. Set $a_{v,1} = 3, a_{v,2} = 2, a_{v,3} = a_{v,4} = 1$, and $g_{w,1}(1) = 1, g_{w,1}(2) = g_{w,2}(1) = 2, g_{w,1}(3) = g_{w,2}(2) = 3, f_{u,1}(1) = f_{u,3}(1) = f_{u,4}(1) = g_{w,3}(1) = g_{w,4}(1) = 4, f_{u,1}(2) = f_{u,1}(3) = f_{u,2}(1) = f_{u,2}(2) = 5$.

According to Lemma 4.9(1), $\mathcal{V}_\emptyset(G_v) = R_{v,1} = \text{MIN}_v\{Z_{u,f_{u,1}(1)} \cup Z_{w,g_{w,1}(1)}, Z_{u,f_{u,1}(2)} \cup Z_{w,g_{w,1}(2)}, \ldots, Z_{u,f_{u,1}(a_{v,1})} \cup Z_{w,g_{w,1}(a_{v,1})}\} = \text{MIN}_v\{Z_{u,4} \cup Z_{w,1}, Z_{u,5} \cup Z_{w,2}, Z_{u,5} \cup Z_{w,3}\}$, $\mathcal{V}^2_{S_v}(G_v) = R_{v,2} = \text{MIN}_v\{Z_{u,f_{u,2}(1)} \cup Z_{w,g_{w,2}(1)}, Z_{u,f_{u,2}(2)} \cup Z_{w,g_{w,2}(2)}, \ldots, Z_{u,f_{u,2}(a_{v,2})} \cup Z_{w,g_{w,2}(a_{v,2})}\} = \text{MIN}_v\{Z_{u,5} \cup Z_{w,2}, Z_{u,5} \cup Z_{w,3}\}$, $\mathcal{V}^3_{S_v}(G_v) = R_{v,3} = \text{MIN}_v\{Z_{u,f_{u,3}(1)} \cup Z_{w,g_{w,3}(1)}, Z_{u,f_{u,3}(2)} \cup Z_{w,g_{w,3}(2)}, \ldots, Z_{u,f_{u,3}(a_{v,3})} \cup Z_{w,g_{w,3}(a_{v,3})}\} = Z_{u,4} \cup Z_{w,4}$, and $S_v = R_{v,4} = Z_{u,4} \cup Z_{w,4}$.

CASE 2: G_w is disconnected. In this case, $\mathcal{V}_\emptyset(G) = \text{MIN}_v\{S_1 \cup conn(G_2), S_1 \cup \mathcal{V}_\emptyset(G_2), \mathcal{V}^2_{S_2}(G_2), \mathcal{V}^3_{S_2}(G_2)\} = \text{MIN}_v\{S_1, S_1 \cup \mathcal{V}_\emptyset(G_2), \mathcal{V}^2_{S_2}(G_2), \mathcal{V}^3_{S_2}(G_2)\}$ because $conn(G_2) = \emptyset$. Set $a_{v,1} = 4, a_{v,2} = 2, a_{v,3} = a_{v,4} = 1$, and $g_{w,1}(2) = 1, g_{w,1}(3) = g_{w,2}(1) = 2, g_{w,1}(4) = g_{w,2}(2) = 3, f_{u,1}(1) = f_{u,1}(2) = f_{u,3}(1) = g_{w,3}(1) = f_{u,4}(1) = g_{w,4}(1) = 4, f_{u,1}(4) = f_{u,2}(1) = f_{u,2}(2) = f_{u,3}(1) = g_{w,1}(1) = 5$.

Then, $\mathcal{V}_\emptyset(G_v) = R_{v,1} = \text{MIN}_v\{Z_{u,4} \cup Z_{w,5}, Z_{u,4} \cup Z_{w,1}, Z_{u,5} \cup Z_{w,2}, Z_{u,5} \cup Z_{w,3}\}$, $\mathcal{V}^2_{S_v}(G_v) = R_{v,2} = \text{MIN}_v\{Z_{u,5} \cup Z_{w,2}, Z_{u,5} \cup Z_{w,3}\}$, $\mathcal{V}^3_{S_v}(G_v) = R_{v,3} = \{Z_{u,4} \cup Z_{w,4}\}$, and $S_v = R_{v,4} = Z_{u,4} \cup Z_{w,4}$.

(S4) For each leaf l corresponding to a primitive distance-hereditary graph $(\{v\}, \emptyset)$, let $R_l = \langle R_{l,1}, R_{l,2}, R_{l,3}, R_{1,4}\rangle = \langle \mathcal{V}_\emptyset(G_l), \mathcal{V}^2_{S_l}(G_l), \mathcal{V}^3_{S_l}(G_l), S_l\rangle = \langle \emptyset, inf, inf, \{v\}\rangle$.

Since (S1) can be implemented in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM by utilizing the binary tree contraction and the other steps can be implemented within the desired complexities, the problem is a $(2, 4, \text{MIN}_v)$-regular problem. □

4.4. The independent domination problem. We say that in a graph $G = (V, E)$, a subset P of V dominates a subset Q of V if every vertex of Q is either in P or adjacent to a vertex in P. A *dominating set* of a graph $G = (V, E)$ is a subset of V that dominates V. A dominating set is *independent* if the subgraph induced by this set has no edge. The *minimum independent domination problem* $\mathcal{ID}$ is to find a minimum cardinality independent dominating set of the given graph. A previous known sequential result of this problem on distance-hereditary graphs can be found in [6]. Another related work can be found in [5]. For a primitive distance-hereditary graph $G = (\{v\}, \emptyset)$, $\mathcal{ID}_\emptyset(G)$ and $\mathcal{ID}_S(G)$ both equal $\{v\}$, $\mathcal{ID}_\emptyset(G[V \setminus S]) = \emptyset$, and $\mathcal{ID}_\emptyset(G, G[V \setminus S]) = inf$.

THEOREM 4.13.

1. *In the case of* $G = G_1 \otimes G_2$,
   - $\mathcal{ID}_\emptyset(G) = \text{MIN}_v\{\mathcal{ID}_{S_1}(G_1) \cup \mathcal{ID}_\emptyset(G_2[V_2 \setminus S_2]), \mathcal{ID}_{S_2}(G_2) \cup \mathcal{ID}_\emptyset(G_1[V_1 \setminus S_1]), \mathcal{ID}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{ID}_\emptyset(G_2, G_2[V_2 \setminus S_2])\}$;
   - $\mathcal{ID}_S(G) = \text{MIN}_v\{\mathcal{ID}_{S_1}(G_1) \cup \mathcal{ID}_\emptyset(G_2[V_2 \setminus S_2]), \mathcal{ID}_{S_2}(G_2) \cup \mathcal{ID}_\emptyset(G_1[V_1 \setminus S_1])\}$;
   - $\mathcal{ID}_\emptyset(G[V \setminus S]) = \mathcal{ID}_\emptyset(G_1[V_1 \setminus S_1]) \cup \mathcal{ID}_\emptyset(G_2[V_2 \setminus S_2])$;
   - $\mathcal{ID}_\emptyset(G, G[V \setminus S]) = \mathcal{ID}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{ID}_\emptyset(G_2, G_2[V_2 \setminus S_2])$.
2. *In the case of* $G = G_1 \oplus G_2$,
   - $\mathcal{ID}_\emptyset(G) = \text{MIN}_v\{\mathcal{ID}_{S_1}(G_1) \cup \mathcal{ID}_\emptyset(G_2[V_2 \setminus S_2]), \mathcal{ID}_{S_2}(G_2) \cup \mathcal{ID}_\emptyset(G_1[V_1 \setminus S_1]), \mathcal{ID}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{ID}_\emptyset(G_2, G_2[V_2 \setminus S_2])\}$;

- $\mathcal{ID}_S(G) = \mathcal{ID}_{S_1}(G_1) \cup \mathcal{ID}_\emptyset(G_2[V_2 \setminus S_2])$;
- $\mathcal{ID}_\emptyset(G[V \setminus S]) = \mathcal{ID}_\emptyset(G_1[V_1 \setminus S_1]) \cup \mathcal{ID}_\emptyset(G_2)$;
- $\mathcal{ID}_\emptyset(G, G[V \setminus S]) = \text{MIN}_v\{\mathcal{ID}_\emptyset(G_1[V_1 \setminus S_1]) \cup \mathcal{ID}_{S_2}(G_2), \mathcal{ID}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{ID}_\emptyset(G_2, G_2[V_2 \setminus S_2])\}$.

3. *In the case of* $G = G_1 \odot G_2$,
   - $\mathcal{ID}_\emptyset(G) = \mathcal{ID}_\emptyset(G_1) \cup \mathcal{ID}_\emptyset(G_2)$;
   - $\mathcal{ID}_S(G) = \text{MIN}_v\{\mathcal{ID}_{S_1}(G_1) \cup \mathcal{ID}_\emptyset(G_2), \mathcal{ID}_{S_2}(G_2) \cup \mathcal{ID}_\emptyset(G_1)\}$;
   - $\mathcal{ID}_\emptyset(G[V \setminus S]) = \mathcal{ID}_\emptyset(G_1[V_1 \setminus S_1]) \cup \mathcal{ID}_\emptyset(G_2[V_2 \setminus S_2])$;
   - $\mathcal{ID}_\emptyset(G, G[V \setminus S]) = \mathcal{ID}_\emptyset(G_1, G_1[V_1 \setminus S_1]) \cup \mathcal{ID}_\emptyset(G_2, G_2[V_2 \setminus S_2])$.

As with the method used in the previous problems, we have the following result.

THEOREM 4.14. *The independent domination problem is a* $(0, 4, \text{MIN}_v)$*-regular problem on distance-hereditary graphs.*

4.5. The domination problem. The *minimum dominating set problem* $\mathcal{D}$ aims at finding a dominating set in the input graph with the minimum cardinality. A related work on distance-hereditary graph can be found in [5]. For a problem $\mathcal{P}_X(G, H)$, $X = \emptyset$ or $X = S$, used in this section, we relax the constraint that H is restricted to be a subgraph of G; i.e., the desired dominating set of G is contained in H, and H may not be a subgraph of G. For a primitive distance-hereditary graph $G = (\{v\}, \emptyset)$, $\mathcal{D}_\emptyset(G), \mathcal{D}_S(G)$ and $\mathcal{D}_S(G[V \setminus S], G)$ are all equal to $\{v\}$, and $\mathcal{D}_\emptyset(G[V \setminus S], G) = \emptyset$.

LEMMA 4.15. *Assume that* $G = G_1 \otimes G_2$.

1. *If* $S_1 = V_1$ *and* $S_2 \neq V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \text{MIN}_v\{\mathcal{D}_{S_2}(G_2), \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2) \cup \{u\}, \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)\}$, *where* $u \in V_1$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)$;
   - $\mathcal{D}_S(G) = \text{MIN}_v\{\mathcal{D}_{S_2}(G_2), \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2) \cup \{u\}, \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)\}$, *where* $u \in V_1$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2)$.
2. *If* $S_1 \neq V_1$ *and* $S_2 = V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \text{MIN}_v\{\mathcal{D}_{S_1}(G_1), \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \{u\}, \mathcal{D}_\emptyset(G_2) \cup \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1)\}$, *where* $u \in V_2$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1)$;
   - $\mathcal{D}_S(G) = \text{MIN}_v\{\mathcal{D}_{S_1}(G_1), \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \{u\}, \mathcal{D}_\emptyset(G_2) \cup \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1)\}$, *where* $u \in V_2$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1)$.
3. *If* $S_1 = V_1$ *and* $S_2 = V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \text{MIN}_v\{\mathcal{D}_\emptyset(G_1), \mathcal{D}_\emptyset(G_2), \{u, w\}\}$, *where* $u \in V_1$ *and* $w \in V_2$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \emptyset$;
   - $\mathcal{D}_S(G) = \text{MIN}_v\{\mathcal{D}_\emptyset(G_1), \mathcal{D}_\emptyset(G_2), \{u, w\}\}$, *where* $u \in V_1$ *and* $w \in V_2$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \{u\}$, *where* $u \in V_1 \cup V_2$.
4. *If* $S_1 \neq V_1$ *and* $S_2 \neq V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \text{MIN}_v\{\mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2), \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2), \mathcal{D}_{S_1}(G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2), \mathcal{D}_{S_2}(G_2) \cup \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1)\}$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)$;
   - $\mathcal{D}_S(G) = \text{MIN}_v\{\mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2), \mathcal{D}_{S_1}(G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2), \mathcal{D}_{S_2}(G_2) \cup \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1)\}$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \text{MIN}_v\{\mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2), \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2) \cup \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1)\}$.

LEMMA 4.16. *Assume that* $G = G_1 \oplus G_2$.

1. *If* $S_1 = V_1$ *and* $S_2 \neq V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \text{MIN}_v\{\mathcal{D}_{S_2}(G_2), \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2) \cup \{u\}, \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)\}$, *where* $u \in V_1$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \text{MIN}_v\{\mathcal{D}_\emptyset(G_2), \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2) \cup \{u\}\}$, *where* $u \in V_1$;
   - $\mathcal{D}_S(G) = \text{MIN}_v\{\mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2) \cup \{u\}, \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)\}$, *where* $u \in V_1$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2) \cup \{u\}$, *where* $u \in V_1$.
2. *If* $S_1 \neq V_1$ *and* $S_2 = V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \text{MIN}_v\{\mathcal{D}_{S_1}(G_1), \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \{w\}, \mathcal{D}_\emptyset(G_2) \cup \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1)\}$, *where* $w \in V_2$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \text{MIN}_v\{\mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_\emptyset(G_2), \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1)\}$;
   - $\mathcal{D}_S(G) = \text{MIN}_v\{\mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \{w\}, \mathcal{D}_{S_1}(G_1)\}$, *where* $w \in V_2$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1)$.
3. *If* $S_1 = V_1$ *and* $S_2 = V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \text{MIN}_v\{\mathcal{D}_\emptyset(G_1), \mathcal{D}_\emptyset(G_2), \{u, w\}\}$, *where* $u \in V_1$ *and* $w \in V_2$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \{u\}$, *where* $u \in V_1$;
   - $\mathcal{D}_S(G) = \text{MIN}_v\{\mathcal{D}_\emptyset(G_1), \{u, w\}\}$, *where* $u \in V_1$ *and* $w \in V_2$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \{u\}$, *where* $u \in V_1$.
4. *If* $S_1 \neq V_1$ *and* $S_2 \neq V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \text{MIN}_v\{\mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2), \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2), \mathcal{D}_{S_1}(G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2), \mathcal{D}_{S_2}(G_2) \cup \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1)\}$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \text{MIN}_v\{\mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_\emptyset(G_2), \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)\}$;
   - $\mathcal{D}_S(G) = \text{MIN}_v\{\mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2), \mathcal{D}_{S_1}(G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)\}$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)$.

LEMMA 4.17. *Assume that* $G = G_1 \odot G_2$.

1. *If* $S_1 = V_1$ *and* $S_2 \neq V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2)$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)$;
   - $\mathcal{D}_S(G) = \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2)$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2)$.
2. *If* $S_1 \neq V_1$ *and* $S_2 = V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2)$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1)$;
   - $\mathcal{D}_S(G) = \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2)$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1)$.
3. *If* $S_1 = V_1$ *and* $S_2 = V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2)$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \emptyset$;
   - $\mathcal{D}_S(G) = \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2)$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \{u\}$, *where* $u \in V_1 \cup V_2$.
4. *If* $S_1 \neq V_1$ *and* $S_2 \neq V_2$, *then*
   - $\mathcal{D}_\emptyset(G) = \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_\emptyset(G_2)$;
   - $\mathcal{D}_\emptyset(G[V \setminus S], G) = \mathcal{D}_\emptyset(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2)\}$;
   - $\mathcal{D}_S(G) = \text{MIN}_v\{\mathcal{D}_{S_1}(G_1) \cup \mathcal{D}_\emptyset(G_2), \mathcal{D}_\emptyset(G_1) \cup \mathcal{D}_{S_2}(G_2)\}$;
   - $\mathcal{D}_S(G[V \setminus S], G) = \text{MIN}_v\{\mathcal{D}_{S_1}(G_1[V_1 \setminus S_1], G_1) \cup \mathcal{D}_\emptyset(G_2[V_2 \setminus S_2], G_2),$

$\mathcal{D}_{S_2}(G_2[V_2 \setminus S_2], G_2) \cup \mathcal{D}_{\emptyset}(G_1[V_1 \setminus S_1], G_1)\}$.

THEOREM 4.18. *The domination problem is a* $(2, 4, \text{MIN}_v)$*-regular problem on distance-hereditary graphs.*

Proof. A corresponding $(2, 4, \text{MIN}_v)$-subgraph generating tree can be constructed by the following steps:
(S1) For each node $v \in V(\mathcal{D}_G)$, determine whether $S_v = V(G_v)$.
(S2) For each node $v \in V(\mathcal{D}_G)$, set $A_v = \langle y, \emptyset \rangle$, where $y \in S_v$.
(S3) For each internal node v, set $a_{v,1}, \ldots, a_{v,4}$ and construct corresponding functions, according to Lemmas 4.15–4.17. The details are similar to those in the proofs of Theorems 4.2 and 4.12.
(S4) For each leaf l corresponding to a primitive distance-hereditary graph $(\{v\}, \emptyset)$, set four target subgraphs of l to be $R_l = \langle \mathcal{D}_{\emptyset}(G_l), \mathcal{D}_{\emptyset}(G_l[V_l \setminus S_l], G_l), \mathcal{D}_{S_l}(G_l), \mathcal{D}_{S_l}(G_l[V_l \setminus S_l], G_l) \rangle = \langle \{v\}, \emptyset, \{v\}, \{v\} \rangle$.
Clearly, the above reduction scheme can be implemented with the desired complexities. Therefore, the desired problem is a $(2, 4, \text{MIN}_v)$-regular problem. □

5. Parallel constructing a decomposition tree. A parallel algorithm to construct a decomposition tree of a distance-hereditary graph is presented in this section.

5.1. Previously known properties of distance-hereditary graphs. For two arbitrary vertices u and v in a given graph H, let $dist_H(u, v)$ be the length of a shortest path between u and v in H. Given a vertex u in a connected graph $G = (V, E)$, the *hanging* of G rooted at u, denoted by h_u, is the collection of sets $L_0(u), L_1(u), \ldots, L_t(u)$ (or simply $L_0, L_1, \ldots, L_t$ without ambiguity), where $t = \max_{v \in V} dist_G(u, v)$ and $L_i(u) = \{v \in V |\ dist_G(u, v) = i\}$ for $0 \le i \le t$. For any vertex $v \in L_i$ and any vertex set $S \subseteq L_i$, $1 \le i \le t$, let $N'(v) = N(v) \cap L_{i-1}$ and $N'(S) = N(S) \cap L_{i-1}$. Any two vertices $x, y \in L_i$ $(1 \le i \le t-1)$ are said to be *tied* if x and y have a common neighbor in L_{i+1}.

A vertex subset S is *homogeneous* in a graph $G = (V, E)$ if every vertex in $V \setminus S$ is adjacent to either all or none of the vertices of S. We call a family of subsets *arboreal* if every two subsets of the family are either disjoint or comparable (by set inclusion). For a hanging $h_u = (L_0, L_1, \ldots, L_t)$, Hammer and Maffray [15] defined an equivalence relation $\equiv_i$ between vertices of L_i by $x \equiv_i y$, which means x and y are in the same connected component of L_i or x and y are tied. Let $\equiv_a$ be defined on $V(G)$ by $x \equiv_a y$, which means $x \equiv_i y$ for some i.

LEMMA 5.1 (see [2, 11, 15]). *Let h_u be the hanging of G rooted at u and let $R_1, R_2, \ldots, R_r$ be the equivalence classes with respect to h_u. Then the following are true.*

1. *For any two vertices x and y in some R_i, $N'(x) = N'(y)$.*
2. *The graph obtained from G by shrinking each R_j into one vertex is a tree rooted at u.*
3. *Each R_j induces a cograph.*
4. *The family $\{N'(R_k) |\ N'(R_k) \subseteq R_i\}$, for $1 \le i \le r$, is an arboreal family of homogeneous subsets of $G[R_i]$.*

A hanging of a distance-hereditary graph is depicted in Figure 5.1.

5.2. One-vertex-extension trees of cographs. A graph is *cograph* [8] if it is either a vertex, the complement of a cograph, or the union of two cographs. The cograph is also called the P_4-free graph which does not contain any induced path of length three [8]. It has been shown that the class of cographs is properly contained in distance-hereditary graphs [15]. A cograph G has a tree representation called *cotree*,

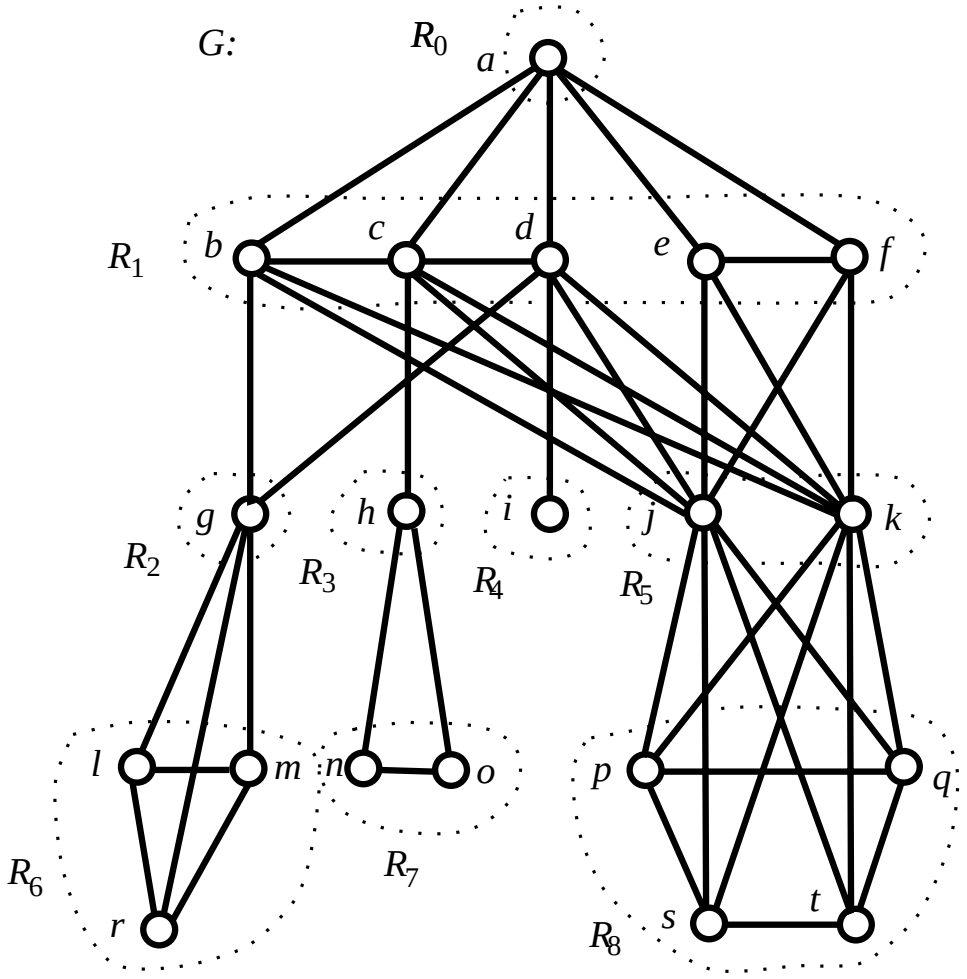

FIG. 5.1. *The hanging h_a of a distance-hereditary graph G. The dotted rings depict a partition of $V(G)$ into nine equivalence classes R_0–R_8.*

denoted by T_G, with the following four properties: (a) the leaves of T_G are the vertices of G; (b) the internal nodes of T_G are labelled with 0 or 1; (c) 0 nodes and 1 nodes alternate along every path staring from the root; (d) two vertices x and y of G are adjacent if and only if the least common ancestor of x and y in T_G is labelled with 1. Cotrees can be utilized to solve the recognition problem and some other subgraph optimization problems on cographs [17, 23]. Figure 5.2 shows a cograph G and its cotree T_G.

Given a tree T, let $leaf(T)$ be the leaves of T.

LEMMA 5.2. *Let u and v be two leaves in a cotree T_G such that $par(u) = par(v)$. If $par(u)$ is labelled with* 1 *(respectively,* 0*), then u and v are true (respectively, false) twins.*

Proof. The proof is straightforward. □

Given a cograph G represented by its cotree T_G, the graph can be reduced to a single vertex by repeatedly merging twins by the following procedure. We arbitrarily

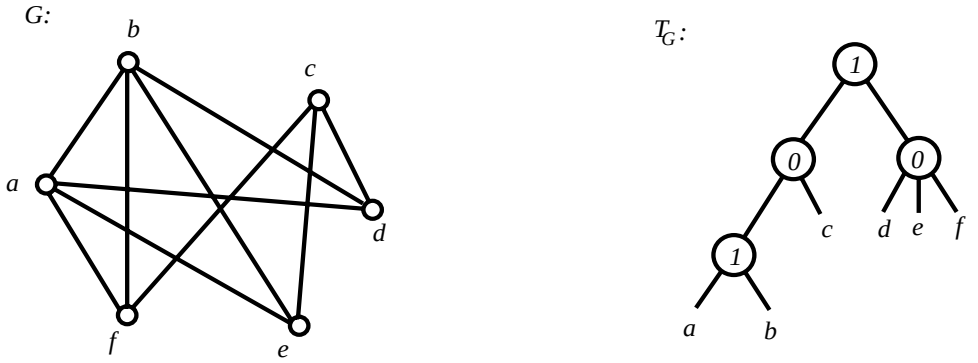

FIG. 5.2. *A cograph and its cotree.*

find two leaves u and v of the current tree with $par(u) = par(v) = w$. By Lemma 5.2, u and v are twins in the current graph. We delete u from the current graph and the current tree. At the same time, we check whether v is the only child of w in the current tree. If so, we delete w from the current tree and let $par(w)$ be the new parent of v when $w \neq r$. The above procedure is repeatedly executed until the current graph contains only one vertex. Clearly, a one-vertex-extension ordering of G can be obtained by reversing the above process. The above discussion leads to the following algorithm.

Algorithm Tree_1

INPUT: A cograph G.

OUTPUT: A one-vertex-extension tree of G.

Step 1: Construct a cotree T_G. Assume that r is the root of T_G.

Step 2: Order the leaves of T_G from 1 to $k = |leaf(T_G)|$. Let $order(v)$ be the resulting order associated with $v \in leaf(T_G)$.

Step 3: Assign a label to each $u \in T_G$:
Find the vertex $v \in leaf(T_G(u))$ such that $order(v) = max\{order(w)|\ w \in leaf(T_G(u))\}$. Let $label(u) = v$.

Step 4: For each $v \in V(G)$, compute $level(v) = min\{dist_{T_G}(x, r)|\ label(x) = v\}$.

Step 5: Construct a tree $\mathcal{E}_G$:
5-1. Let $label(r)$ be the root of $\mathcal{E}_G$.
5-2. For each nonroot node $v \in T_G$, let $par(label(v)) = label(par(v))$ if $label(v) \neq label(par(v))$.
5-3. Label edge $(label(v), label(par(v)))$ as T (respectively, F) if $par(v)$ is a 1 (respectively, 0) node.

Step 6: Order the children of each nonleaf vertex $v \in \mathcal{E}_G$:
Assume that $v_1, v_2, \ldots, v_p$ are p children of v. Order them by $v_{i_1} < v_{i_2} < \cdots < v_{i_p}$ if $level(v_{i_1}) \leq level(v_{i_2}) \leq \cdots \leq level(v_{i_p})$, where $1 \leq i_j \leq p$. The resulting tree is a one-vertex-extension tree of G.

An example of executing Algorithm Tree_1 is shown in Figure 5.3. In Figure 5.3(a), the numbers associated with the leaves form an order determined after Step 2. The bold letters associated with internal nodes v represent $label(v)$. In Figure 5.3(b), a one-vertex-extension tree is generated after Steps 4–6.

The correctness follows from the statements preceding the algorithm. The time-processor complexity of Algorithm Tree_1 is analyzed below. In Step 1, T_G can be

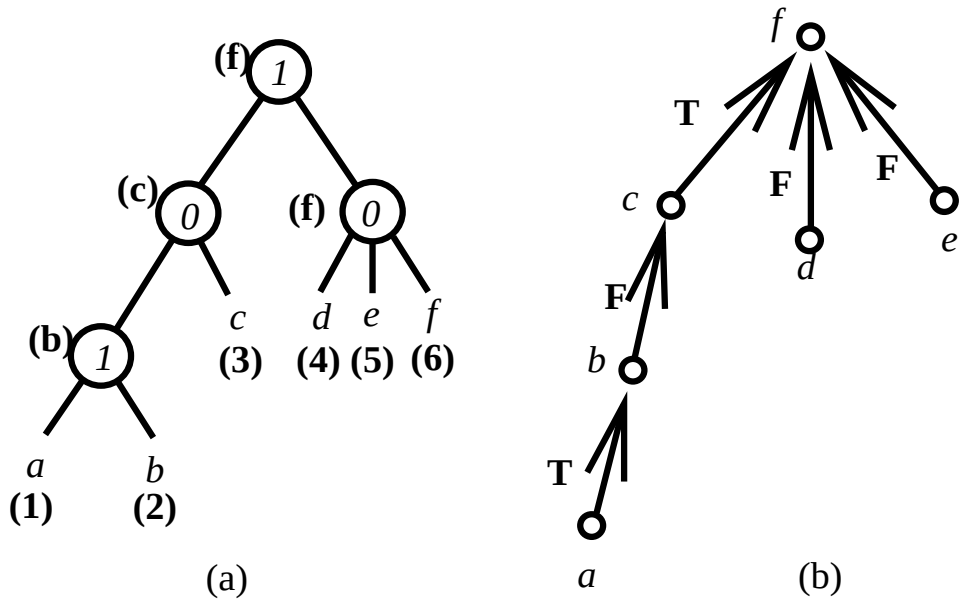

FIG. 5.3. *A one-vertex-extension tree shown in* (b) *is obtained from the given cotree shown in* (a).

constructed in $O(\log^2 n)$ time using $O(n+m)$ processors on a CREW PRAM [10]. As with the aid of the Euler-tour, the prefix-sum and the tree contraction techniques [21], Steps 2 and 3 can be implemented in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM. Step 4 can be done within the above complexities using the Euler-tour technique together with the result of finding minimum value [21]. Step 5 can be done in $O(1)$ time using $O(n)$ processors. By utilizing Cole's parallel merge sort [7], Step 6 can be implemented in $O(\log n)$ time using $O(n)$ processors on an EREW PRAM. Therefore, we have the following theorem.

THEOREM 5.3. *Algorithm Tree_1 correctly constructs a one-vertex-extension tree for a cograph in* $O(\log^2 n)$ *time using* $O(n+m)$ *processors on a CREW PRAM.*

5.3. One-vertex-extension trees of distance-hereditary graphs. Throughout this section, G is used to denote a distance-hereditary graph unless stated otherwise.

Let R be an equivalence class of G with respect to a hanging h_u. We call $\Gamma_R = \{Y \subset R|$ there is an equivalence class R' with $N'(R') = Y\}$ the *upper neighborhood system in* R and call each $S \in \Gamma_R$, where $S = N'(R')$, the *upper neighborhood of* R'. By Lemma 5.1, Γ_R is an arboreal family of homogeneous subsets of R. We define a partial order $\preceq$ between two different sets Y_p and Y_q in Γ_R with $Y_p \preceq Y_q \Leftrightarrow Y_p \subset Y_q$. According to the partial order $\preceq$ defined on Γ_R, let $\mathcal{U}_R = \{Y_i|\ Y_i \not\subseteq Y_k$, for all $Y_k \in \Gamma_R$ and $k \neq i\}$; that is, $\mathcal{U}_R$ is the set of those maximal elements of $(\preceq, \Gamma_R)$. We call $\mathcal{U}_R$ the *maximal upper neighborhoods in* R. For a set Y that is the upper neighborhood of some equivalence class, we can also define Γ_Y and $\mathcal{U}_Y$ similarly. In what follows, the notation R is referred to as an equivalence class or an upper neighborhood of some equivalence class if it is not specified.

LEMMA 5.4. *Let* $\mathcal{U}_R = \{Q_1, Q_2, \ldots, Q_k\}$ *and* x_i *be an arbitrary vertex of* Q_i, $1 \leq i \leq k$. *The graph* $G[(R \setminus \cup_{i=1}^{k} Q_i) \cup \{x_1, x_2, \ldots, x_k\}]$ *is a cograph.*

Proof. By the property that every induced subgraph of a P_4-free graph remains P_4-free, the result holds. □

Let $G = (V, E)$ be a cograph and let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_t\}$ be the set consisting of homogeneous sets of G such that $Q_i \cap Q_j = \emptyset$, $1 \leq i, j \leq t$ and $i \neq j$. Also let $G' = G[(V \setminus (\cup_{i=1}^{t} Q_i)) \cup \{x_1, x_2, \ldots, x_t\}]$, where $x_i \in Q_i$. The following procedure can be used to construct a one-vertex-extension tree of G by merging one-vertex-extension trees of G' and $G[Q_i]$'s.

Procedure 1

S1: Construct a one-vertex-extension tree $\mathcal{E}'$ of G'. For each x_i in $\mathcal{E}'$, $1 \leq i \leq t$, let $(c_1^i, c_2^i, \ldots, c_{j_i}^i)$ be the children of x_i.

S2: Construct a one-vertex-extension tree $\mathcal{E}_i$ for each $G[Q_i]$. Let r_i be the root of $\mathcal{E}_i$ and let $(d_1^i, d_2^i, \ldots, d_{l_i}^i)$ be the children of r_i. Rename the vertex x_i in $\mathcal{E}'$ as r_i.

S3: Construct a tree $\mathcal{E}_G$ by identifying each root r_i of $\mathcal{E}_i$ with the vertex r_i in $\mathcal{E}'$, $1 \leq i \leq t$, such that $(c_1^i, c_2^i, \ldots, c_{j_i}^i, d_1^i, d_2^i, \ldots, d_{l_i}^i)$ are the resulting children of r_i in $\mathcal{E}_G$.

LEMMA 5.5. *The tree $\mathcal{E}_G$ constructed in Procedure* 1 *is a one-vertex-extension tree of a cograph G.*

Proof. We show the lemma by induction on $|\mathcal{Q}| = t$. The base case of $t = 0$ holds clearly. Suppose now that $t > 0$. By the proof of Lemma 5.4, the graph $G_1 = G[(V \setminus Q_1) \cup \{x_1\}]$ is a cograph with $t-1$ homogeneous sets $Q_2, Q_3, \ldots, Q_t$. By the induction hypothesis, a one-vertex-extension tree $\mathcal{E}_{G_1}$ can be correctly constructed using Procedure 1. Since Q_1 is a homogeneous set, $N_{G_1}(x_1) = (N_G(y) \setminus Q_1)$ for $y \in Q_1 \setminus \{x_1\}$, and $E(G) = E(G_1) \cup E(G[Q_1]) \cup \{(z, b) |\ z \in Q_1, b \in N_{G_1}(x_1)\}$. By executing S2 of Procedure 1, a one-vertex-extension tree $\mathcal{E}_1$ of Q_1 can be obtained. By S3 of Procedure 1 and the definition of the one-vertex-extension tree, the graph corresponding to $\mathcal{E}_G$ is obtained by connecting $G[Q_1]$ and G_1 through edges $\{(z, b) |\ z \in Q_1, b \in N_{G_1}(x_1)\}$. Hence, $\mathcal{E}_G$ is a one-vertex-extension tree of G. □

For ordered k children $(v_{i_1}, v_{i_2}, \ldots, v_{i_k})$ of a node v_i in $\mathcal{E}_G$, recall that $\mathcal{E}_G(v_{i_j}, v_i)$ is the subtree of $\mathcal{E}_G$ induced by $v_i, v_{i_j}, v_{i_{j+1}}, \ldots, v_{i_k}$ and all descendants of $v_{i_j}, v_{i_{j+1}}, \ldots, v_{i_k}$.

DEFINITION 5.6. *Let R be an equivalence class with respect to a hanging. A one-vertex-extension tree $\mathcal{E}_{G[R]}$ is* canonical *if for each $Q \in \Gamma_R$ there exist a vertex $v_i \in Q$ and one of its children v_{j_i} such that $\mathcal{E}_{G[R]}(v_{j_i}, v_i)$ is a one-vertex-extension tree of $G[Q]$.*

Given R and Γ_R, the following procedure can be used to construct a canonical one-vertex-extension tree of $G[R]$.

Procedure 2

S1: Let $\Gamma_R \cup \{R\} = \{Y_1, Y_2, \ldots, Y_t\}$, and let $\mathcal{U}_{Y_i} = \{Y_{i_1}, Y_{i_2}, \ldots, Y_{i_{l_i}}\}$, where $Y_1 = R$ and $2 \leq i_j \leq t$ for $1 \leq j \leq l_i$. For each $Y_i \in \Gamma_R$ and $|Y_i| > 1$, select a *shrinking vertex* $y_i \in Y_i$.

S2: Let $Y_i' = (Y_i \setminus \cup_{j=1}^{l_i} Y_{i_j}) \cup \{y_{i_1}, y_{i_2}, \ldots, y_{i_{l_i}}\}$. Construct a one-vertex-extension trees $\mathcal{E}_{G[Y_i']}$'s, $1 \leq i \leq t$, using Algorithm Tree_1.

S3: For each $1 \leq i \leq t$, merge trees $\mathcal{E}_{G[Y_i']}$ and $\mathcal{E}_{G[Y_{i_1}']}, \mathcal{E}_{G[Y_{i_2}']}, \ldots, \mathcal{E}_{G[Y_{i_{l_i}}']}$ using Procedure 1.

LEMMA 5.7. *The tree constructed using Procedure* 2 *is a canonical one-vertex-extension tree for $G[Y_1] = G[R]$.*

Proof. The proof is by induction on $|\Gamma_{Y_1}|$. The base case of $\Gamma_{Y_1} = \emptyset$ trivially holds. Now we consider $|\Gamma_{Y_1}| > 0$. By the induction hypothesis, the canonical one-

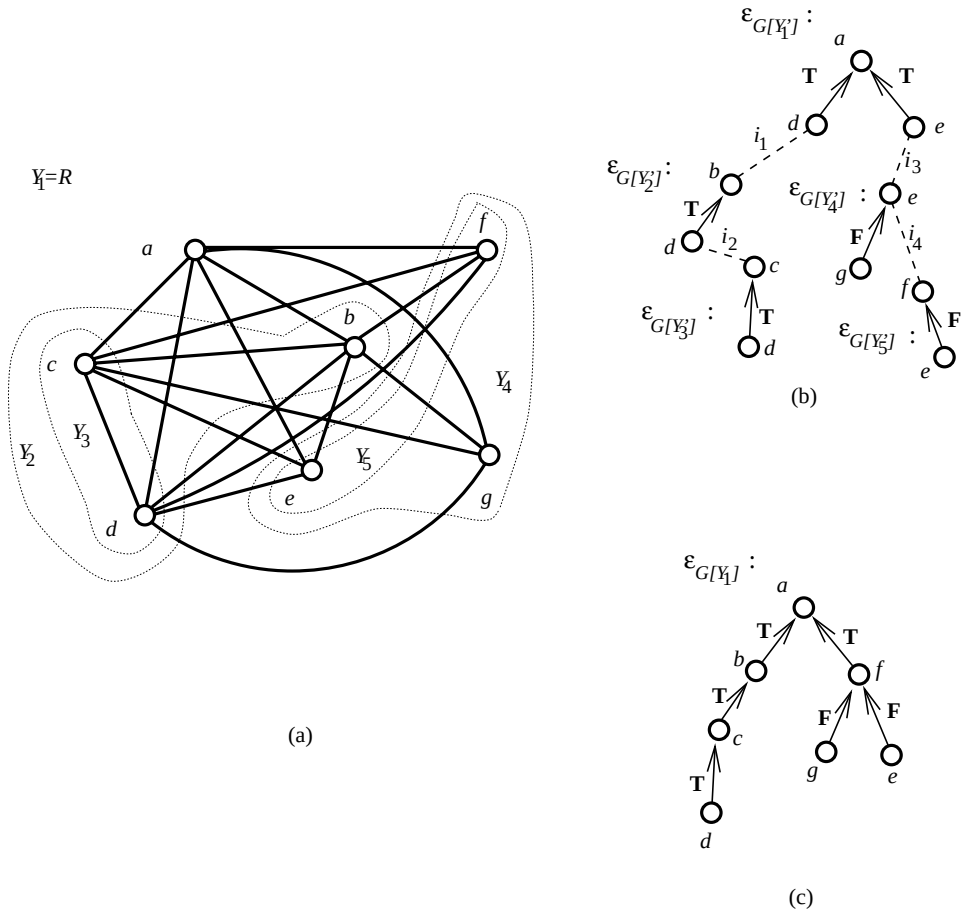

FIG. 5.4. *An example of constructing a canonical one-vertex-extension tree using Procedure* 2. *The dotted lines shown in* (b) *represent identifying operations.*

vertex-extension trees $\mathcal{E}_{G[Y_{1_j}]}$ of $G[Y_{1_j}]$, $1 \leq j \leq l_1$, can be correctly constructed using Procedure 2. By Lemma 5.1(4), Y_{1_j} is a homogeneous set of $G[Y_1]$. By the definition of $\mathcal{U}_{Y_1}$, $Y_{1_p} \cap Y_{1_q} = \emptyset$, $1 \leq p, q \leq l_1$, and $p \neq q$. The operations used to merge $\mathcal{E}_{G[Y_1{}']}$ and $\mathcal{E}_{G[Y_{1_j}]}$'s are based on Procedure 1. Hence, the resulting tree is a one-vertex-extension tree of $G[Y_1]$. Moreover, the canonical property holds from the construction. □

Figure 5.4 shows an example of generating a canonical one-vertex-extension tree using Procedure 2. Consider $Y_1 = R = \{a, b, c, d, e, f, g\}, Y_2 = \{b, c, d\}, Y_3 = \{c, d\}, Y_4 = \{e, f, g\}$, and $Y_5 = \{e, f\}$ (see Figure 5.4(a)). Note that $\mathcal{U}_{Y_1} = \{Y_2, Y_4\}, \mathcal{U}_{Y_2} = \{Y_3\}$, and $\mathcal{U}_{Y_4} = \{Y_5\}$. Let $y_2 = d, y_3 = d, y_4 = e$, and $y_5 = e$. In Figure 5.4(b), trees $\mathcal{E}_{G[Y_i{}']}$ are constructed after S2. The identifying operations shown as dotted lines i_1–i_4 are then executed in S3. Note that i_1 and i_2 are executed after the labels of d in $\mathcal{E}_{G[Y_1{}']}$ and d in $\mathcal{E}_{G[Y_2{}']}$ have been changed to b and c, respectively. Also note that the resulting tree can be constructed correctly despite the operations i_3 and i_4 involving the vertex which is the root of $\mathcal{E}_{G[Y_4{}']}$ and also the shrinking vertex of Y_5. Figure 5.4(c) shows the tree produced after executing Procedure 2.

We now present an algorithm to construct a one-vertex-extension tree of a distance-hereditary graph.

Algorithm Tree_2

INPUT: A distance-hereditary graph G.

OUTPUT: A one-vertex-extension tree $\mathcal{E}_G$.

Step 1: Build a hanging h_u and compute the equivalence classes with respect to h_u.

Step 2: For each equivalence class R, compute Γ_R.

Step 3: For each equivalence class R, generate a canonical one-vertex-extension tree $\mathcal{E}_{G[R]}$ using Procedure 2.

Step 4: For each equivalence class R, let R' be the equivalence class with $N'(R) \subseteq R'$. Find the subtree $\mathcal{E}_{G[N'(R)]}$ in $\mathcal{E}_{G[R']}$, which is a one-vertex-extension tree of $G[N'(R)]$. Let $root(\mathcal{E}_{G[N'(R)]})$ be the root of $\mathcal{E}_{G[N'(R)]}$ and let $(c_1, c_2, \ldots, c_{l_{N'(R)}})$ be the children of $root(\mathcal{E}_{G[N'(R)]})$ in $\mathcal{E}_{G[R']}$. Construct $\mathcal{E}_G$ as follows. Let the root of $\mathcal{E}_{G[R]}$ be a new child of $root(\mathcal{E}_{G[N'(R)]})$ which is located between c_{i_R} and c_{i_R+1} for some $1 \leq i_R \leq l_{N'(R)}-1$ such that $\mathcal{E}_{G[R']}(c_{i_R+1}, root(\mathcal{E}_{G[N'(R)]}))$ equals $\mathcal{E}_{G[N'(R)]}$. The edge $(root(\mathcal{E}_{G[R]}), root(\mathcal{E}_{G[N'(R)]}))$ is labelled with "P."

Figure 5.5 shows the construction of a one-vertex-extension tree of the graph shown in Figure 5.1. The nine canonical one-vertex-extension trees $\mathcal{E}_{G[R_i]}$'s for $0 \leq i \leq 8$ are generated in Step 3. The dotted lines represent those operations executed in Step 4.

Recall that shrinking each equivalence class with respect to the given hanging h_u forms a tree (see Lemma 5.1(2)). We use T_{h_u} to denote such a tree. For each equivalence class R, let ν_R be the node representing R in T_{h_u}. Let $\psi(R) = \{Q |\ \nu_Q \in V(T_{h_u}(\nu_R))\}$ and let $\psi'(R) = \bigcup_{X \in \psi(R)} X$.

LEMMA 5.8. *Algorithm Tree_2 correctly constructs a one-vertex-extension tree of* $G[\psi'(R)]$.

Proof. The proof is by induction on $|\psi(R)|$. The base case of $\psi(R) = \{R\}$ trivially holds. Suppose now that $|\psi(R)| = t > 1$. Let $R_1, R_2, \ldots, R_r$ be the equivalence classes with $N'(R_i) \subseteq R$. After Step 3, a canonical one-vertex-extension tree $\mathcal{E}_{G[R]}$ can be constructed. Note that $|\psi(R_i)| < t$ for all $1 \leq i \leq r$. By the induction hypothesis, the one-vertex-extension trees $\mathcal{E}_{G[\psi'(R_i)]}$'s can be correctly constructed using Algorithm Tree_2. After Step 4, the graph corresponding to $\mathcal{E}_{G[\psi'(R)]}$ can be obtained from $G[R]$ (corresponding to $\mathcal{E}_{G[R]}$) and $G[\psi'(R_i)]$ (corresponding to $\mathcal{E}_{G[\psi'(R_i)]}$), $1 \leq i \leq r$, by making R_i and $N'(R_i)$ form a join. According to the structure characterization described in Lemma 5.1, the resulting tree is a one-vertex-extension tree of $G[\psi'(R)]$. □

By Lemma 5.8, Algorithm Tree_2 correctly constructs a one-vertex-extension tree of $G[\psi'(\{u\})] = G$, where u is the root of the given hanging.

We now analyze the time-processor complexity. Step 1 and Step 2 can be implemented to run in $O(\log^2 n)$ time using $O(n+m)$ processors on a CREW PRAM [19].

To implement Step 3, we need to implement Steps (S1)–(S3) of Procedure 2. In (S1), given $\Gamma_R \cup \{R\} = \{Y_1, Y_2, \ldots, Y_t\}$, we find $\mathcal{U}_{Y_i}$ in $O(\log |R|)$ time using $O(\sum_{i=1}^{t} |Y_i|)$ processors on an EREW PRAM [19]. Clearly, selecting a shrinking vertex can be done in $O(1)$ time using $O(t)$ processors. Thus (S1) can be implemented in $O(\log |R|)$ time using $O(\sum_{i=1}^{t} |Y_i|)$ processors on an EREW PRAM. The complexities of (S2) are bounded by constructing $\mathcal{E}_{G[Y_i']}$, $1 \leq i \leq t$. By Theorem 5.3, this step can be implemented in $O(\log^2 |R|)$ time using $O(E(G[R]))$ processors on a CREW PRAM. After executing this step, we assume that the children of each node in a given

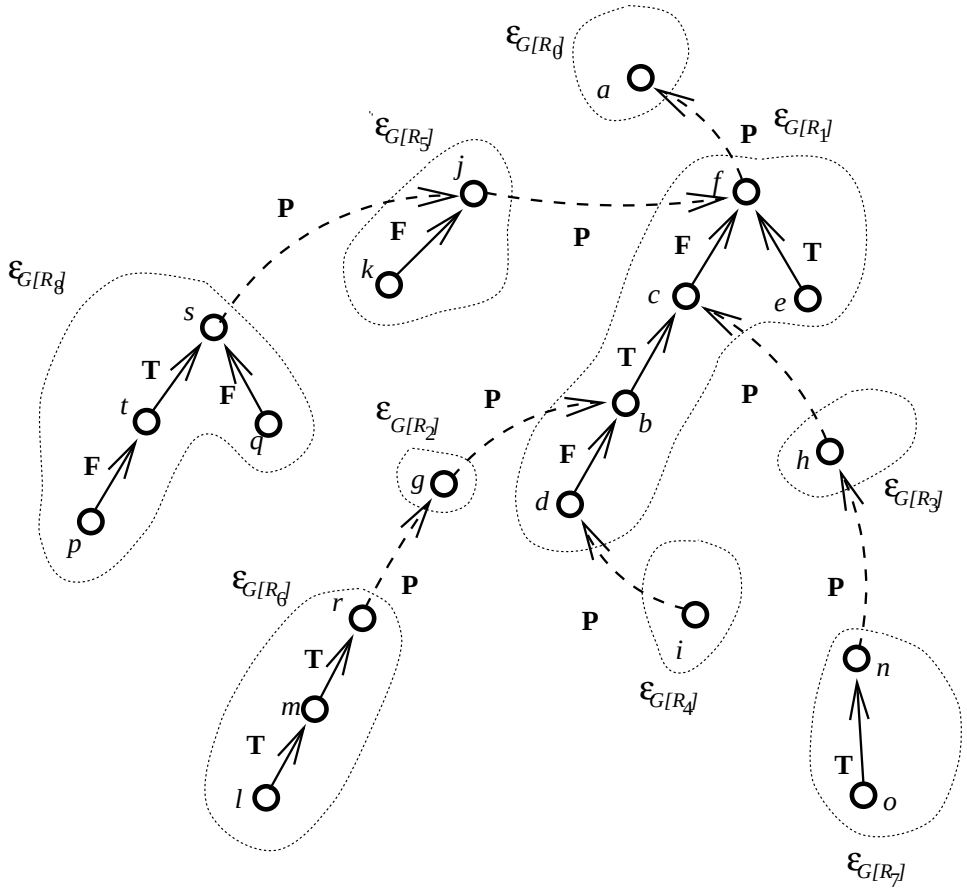

FIG. 5.5. *An example of executing Algorithm Tree_2.*

tree are manipulated using an ordered list. In (S3), we merge desired trees based on the identifying operations in Procedure 1. Those operations can be implemented in $O(\log t)$ time using $O(t)$ processors on a CREW PRAM. By utilizing the list-ranking technique and the prefix-sum technique [21], we maintain the children of each node in the resulting tree through merging lists. Therefore, Step 3 can be implemented within $O(\log^2 n)$ time using $O(n+m)$ processors on a CREW PRAM. Similarly, Step 4 can be implemented with the desired complexities. Then, we have the following theorem.

THEOREM 5.9. *Algorithm Tree_2 correctly constructs a one-vertex-extension tree of a distance-hereditary graph in $O(\log^2 n)$ time using $O(n+m)$ processors on a CREW PRAM.*

5.4. Decomposition trees of distance-hereditary graphs. Throughout this section, we assume that each vertex of G is represented by its corresponding one-vertex-extension order. By Lemma 2.3, the following recursive method can be used to transform a one-vertex-extension tree into a decomposition tree. Let $\mathcal{E}$ be a given one-vertex-extension tree whose root and leftmost child are x and y, respectively. If (y, x) is labelled with T, then we create a $\otimes$-node as the root of a decomposition tree $\mathcal{D}_{G[V(\mathcal{E}(x))]}$. If (y, x) is labelled with P, then we create a $\oplus$-node as the root of

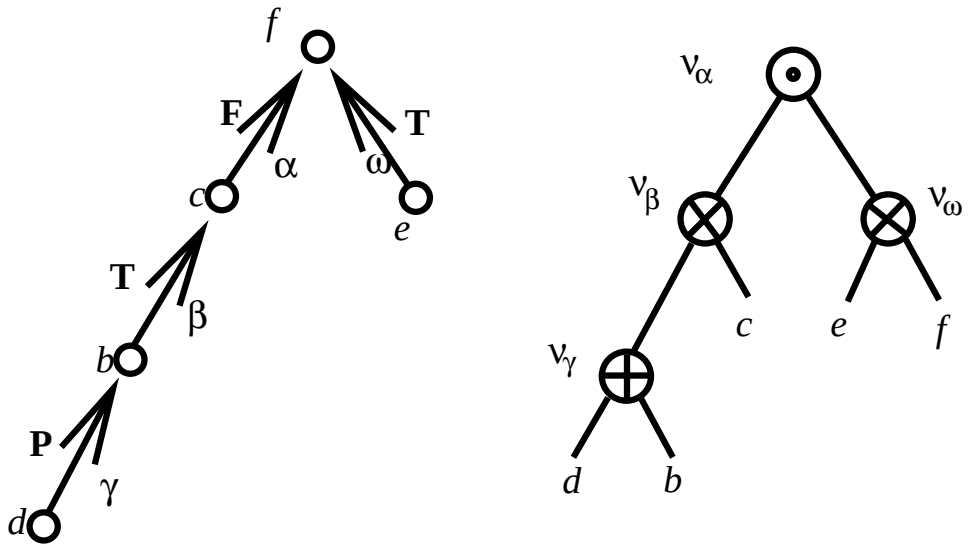

FIG. 5.6. *An example of executing Algorithm Tree_3. The tree shown in the left is an input and that shown in the right is its corresponding output.*

$\mathcal{D}_{G[V(\mathcal{E}(x))]}$. Otherwise, we create a $\odot$-node as the root of $\mathcal{D}_{G[V(\mathcal{E}(x))]}$. After recursively constructing $\mathcal{D}_{G[V(\mathcal{E}(y))]}$ and $\mathcal{D}_{G[V(\mathcal{E}(x))\setminus V(\mathcal{E}(y))]}$), we let the roots of $\mathcal{D}_{G[V(\mathcal{E}(y))]}$ and $\mathcal{D}_{G[V(\mathcal{E}(x))\setminus V(\mathcal{E}(y))]}$ be the left child and the right child of the created node for (y, x), respectively. The above method can be implemented using the following nonrecursive algorithm.

Algorithm Tree_3

INPUT: A one-vertex-extension tree $\mathcal{E}_G$.

OUTPUT: A decomposition tree $\mathcal{D}_G$.

Step 1: For each vertex v in $\mathcal{E}_G$, let $num(v)$ be the one-vertex-extension order associated with v. For each edge $e = (v, par(v))$ in $\mathcal{E}_G$, let $num(e) = num(v)$.

Step 2: For each edge e in $\mathcal{E}_G$, create an internal node ν_e ($\otimes$ or $\oplus$ or $\odot$) for $\mathcal{D}_G$ depending on the label of e.

Step 3: For each node ν_e, where $e = (v, par(v))$, execute the following operations:
(a) If $par(v)$ contains no child w in $V(\mathcal{E}_G)$ such that $num((w, par(v))) > num(e)$, create a node representing $par(v)$ to be the right child of ν_e. Otherwise, find the edge e' next to e. Let the node created for e' be the right child of ν_e.
(b) If v is a leaf in $V(\mathcal{E}_G)$, create a node representing v to be the left child of ν_e. Otherwise, find the edge $e' = (z, v)$ such that $num(z) = min\{num(x)|\ x \in child(v)\}$. Let the node created for e' be the left child of ν_e.

Figure 5.6 shows a one-vertex-extension tree with its corresponding decomposition tree. The nodes $\nu_\alpha, \nu_\beta, \nu_\gamma, \nu_\omega$ are created in Step 2 of Algorithm Tree_3. The left child and the right child of each node ν_e, where e is an edge in $\{\alpha, \beta, \gamma, \omega\}$, are determined in Step 3 of Algorithm Tree_3.

The correctness follows from the statements preceding the algorithm. Based on the data structure maintained in Algorithm Tree_2 and the Euler-tour technique [21], we have the following result.

THEOREM 5.10. *Algorithm Tree_3 correctly transforms a one-vertex-extension tree into a decomposition tree in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM.*

6. Discussion and conclusion. In this paper, we first define the (r, k, Θ)-subgraph generating problem on trees. We solve this problem in $O((rk + k^2)n)$ sequential time, and in $O(k^2(r + k)\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM, where n is the number of nodes of the given tree. We then develop a general problem-solving paradigm used to reduce a class of subgraph optimization problems on distance-hereditary graphs to its corresponding (r, k, Θ)-subgraph generating problems. Using this paradigm, we define a class of (r, k, Θ)-regular problems on distance-hereditary graphs. Let $T_d(|V|, |E|)$ and $P_d(|V|, |E|)$ denote the time complexity and processor complexity required to construct a decomposition tree of a distance-hereditary graph $G = (V, E)$ on a PRAM model M_d. We show that an (r, k, Θ)-regular problem on a distance-hereditary graph $G = (V, E)$ can be solved in sequential $O((rk + k^2)n + m)$ time, and in $O(T_d(n, m) + \log n)$ time using $O(P_d(n, m) + n/\log n)$ processors on M_d. We also show that $T_d(n, m) = O(\log^2 n)$, $P_d(n, m) = O(n + m)$ under a CREW PRAM.

Several fundamental graph problems are shown to be (r, k, Θ)-regular, including the maximum clique problem, the maximum independent set problem, the vertex connectivity problem, the domination problem, and the independent domination problem. Therefore, the above problems can be solved in linear time, and in $O(\log^2 n)$ time using $O(n + m)$ processors on a CREW PRAM. Opposed to less parallel results on distance-hereditary graphs, our method classifies a class of problems on distance-hereditary graphs to be in NC. We believe that more graph problems can be shown to be in (r, k, Θ)-regular class.

We note that Golumbic and Rotics [14] showed that a distance-hereditary graph has clique-width at most three and can be represented by a so called 3-expression. Using this structure, it is shown that a class of problems can be solved in sequential linear time on distance-hereditary graphs if those problems can be represented in monadic second order logic with quantification over vertex sets only (MSOL problems for short) [9]. Note that Bodlaender and Hagerup [4] developed a general parallel algorithm to solve several subgraph optimization problems on special classes of graphs with bounded tree-width. However, the tree-width of distance-hereditary graphs is not bounded. It is hopeful and certainly interesting to see if clique-width can be used similarly to solve subgraph optimization problems in parallel. However, to the best of our knowledge, no such result exists.

In [24], Miller and Teng presented a systemic method for the design of efficient parallel algorithms for the dynamic evaluation of computation trees and/or expressions. Their method involves the use of uniform closure properties of certain classes of unary functions. In this paper, we extend their work by considering k-ary functions. Let D be the power set of some given set and let MIN (respectively, MAX) be the operator defined on a subset of D that returns a set with the minimum (respectively, maximum) cardinality. We show that a class algebraic computation tree over $\{D, \text{MIN}, \text{MAX}, \cup\}$ can be optimally evaluated using a class of k-ary functions which is closed under the composition.

Acknowledgments. The authors are deeply appreciative for the comments and suggestions given by the editor and the two anonymous referees.

REFERENCES

[1] K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka, *A simple parallel tree contraction algorithm*, J. Algorithms, 10 (1989), pp. 287–302.
[2] H. J. Bandelt and H. M. Mulder, *Distance-hereditary graphs,* J. Combin. Theory Ser. B, 41 (1986), pp. 182–208.
[3] C. Berge, *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.
[4] H. L. Bodlaender and T. Hagerup, *Parallel algorithms with optimal speedup for bounded treewidth*, SIAM J. Comput., 27 (1998), pp. 1725–1746.
[5] A. Brandstädt and F. F. Dragan, *A linear time algorithm for connected γ-domination and Steiner tree on distance-hereditary graphs*, Networks, 31 (1998), pp. 177–182.
[6] M. S. Chang, S. Y. Hsieh, and G. H. Chen, *Dynamic programming on distance-hereditary graphs,* in Proceedings of the 7th International Symposium on Algorithms and Computation (ISAAC'97), Lecture Notes in Comput. Sci. 1350, Springer-Verlag, Berlin, 1997, pp. 344–353.
[7] R. Cole, *Parallel merge sort*, SIAM J. Comput., 17 (1988), pp. 770–785.
[8] D. G. Corneil, H. Lerchs, and L. S. Burlingham, *Complement reducible graphs*, Discrete Appl. Math., 3 (1981), pp. 163–174.
[9] B. Courcelle, J. A. Makowsky, and U. Rotics, *Linear time solvable optimization problems on graphs of bounded clique-width*, Theory Comput. Syst., 33 (2000), pp. 125–150.
[10] E. Dahlhaus, *Efficient parallel recognition algorithms of cographs and distance-hereditary graphs*, Discrete Appl. Math., 57 (1995), pp. 29–44.
[11] A. D'atri and M. Moscarini, *Distance-hereditary graphs, Steiner trees, and connected domination*, SIAM J. Comput., 17 (1988), pp. 521–538.
[12] F. F. Dragan, *Dominating cliques in distance-hereditary graphs*, in Algorithm Theory-SWAT'94: 4th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 824, Springer-Verlag, Berlin, 1994, pp. 370–381.
[13] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.
[14] M. C. Golumbic and U. Rotics, *On the clique-width of perfect graph classes*, in Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'99), Lecture Notes in Comput. Sci. 1665, Springer-Verlag, Berlin, 1999, pp. 135–147.
[15] P. L. Hammer and F. Maffray, *Complete separable graphs*, Discrete Appl. Math., 27 (1990), pp. 85–99.
[16] X. He, *Efficient parallel algorithms for solving some tree problems*, in Proceedings of the 24th Allerton Conference on Communication, Control, and Computing, 1986, pp. 777–786.
[17] X. He, *Parallel algorithm for cograph recognition with applications*, J. Algorithms, 15 (1993), pp. 284–313.
[18] E. Howorka, *A characterization of distance-hereditary graphs*, Quart. J. Math. Oxford Ser. (2), 28 (1977), pp. 417–420.
[19] S.-y. Hsieh, C. W. Ho, T.-s. Hsu, M. T. Ko, and G. H. Chen, *Efficient parallel algorithms on distance-hereditary graphs*, Parallel Process. Lett., 9 (1999), pp. 43–52.
[20] S.-y. Hsieh, C. W. Ho, T.-s. Hsu, M. T. Ko, and G. H. Chen, *A faster implementation of a parallel tree contraction scheme and its application on distance-hereditary graphs*, J. Algorithms, 35 (2000), pp. 50–81.
[21] J. Ja'Ja', *An Introduction to Parallel Algorithms*, Addison-Wesley, Reading, MA, 1992.
[22] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared memory machines*, in Handbook of Theoretical Computer Science, North-Holland, Amsterdam, 1990, pp. 869–941.
[23] R. Lin and S. Olariu, *An optimal parallel matching algorithm for cographs*, J. Parallel Distrib. Comput., 22 (1994), pp. 26–36.
[24] G. L. Miller and S. H. Teng, *Tree-based parallel algorithm design*, Algorithmica, 19 (1997), pp. 369–389.
[25] F. Nicolai, *Hamiltonian Problems on Distance-Hereditary Graphs*, Technique report, Gerhard-Mercator University, Duisburg, Germany, 1994.
[26] Y. Shiloach and U. Viskin, *An $O(\log n)$ parallel connectivity algorithm*, J. Algorithms, 3 (1982), pp. 57–63.

[27] H. G. Yeh and G. J. Chang, *Weighted connected domination and Steiner trees in distance-hereditary graphs*, Discrete Appl. Math., 87 (1998), pp. 245–253.
[28] H. G. Yeh and G. J. Chang, *Linear-Time Algorithms for Bipartite Distance-Hereditary Graphs*, manuscript.
[29] H. G. Yeh and G. J. Chang, *The path-partition problem in bipartite distance-hereditary graphs*, Taiwanese J. Math., 2 (1998), pp. 353–360.