Journal of Algorithms **35,** 50–81 (2000)
doi:10.1006/jagm.1999.1064, available online at http://www.idealibrary.com on IDEAL®

A Faster Implementation of a Parallel Tree Contraction Scheme and Its Application on Distance-Hereditary Graphs[1]

Sun-yuan Hsieh[2]

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
E-mail: d3506013@csie.ntu.edu.tw

Chin-Wen Ho[3]

Department of Computer Science and Information Engineering, National Central University, Chung-Li, Taiwan
E-mail: hocw@csie.ncu.edu.tw

Tsan-sheng Hsu[4] and Ming-Tat Ko[4]

Institute of Information Science, Academia Sinica, Taipei, Taiwan
E-mail: tshsu@iis.sinica.edu.tw, mtko@iis.sinica.edu.tw

and

Gen-Huey Chen

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan
E-mail: ghchen@csie.ntu.edu.tw

Received March 6, 1998

We consider a parallel tree contraction scheme which in each contraction phase removes leaves and nodes in the maximal chains. Let $T(n)$ and $P(n)$ denote the time and processor complexity required to compute the all nearest smaller values (ANSV) and the minimum of n values for input elements drawn from the integer

[1] A preliminary version of this paper appeared in "Proceedings of 5th International Symposium on Solving Irregularly Structured Problems in Parallel (IRREGULAR'98)," Lecture Notes in Computer Science, Vol. 1457, pp. 298–309, Springer-Verlag, Berlin, 1998.
[2] Supported in part by Institute of Information Science, Academia Sinica, Taipei, Taiwan.
[3] To whom correspondence should be addressed.
[4] Supported in part by the NSC of Taiwan under Grant 86-2213-E-001-012.

0196-6774/00 $35.00

domain $[1 \ldots n]$. In this paper, we give a faster implementation of the tree contraction scheme which takes $O(\log n \cdot T(n))$ time using $P(n)$ processors on an arbitrary CRCW PRAM. The current best results of $T(n)$ and $P(n)$ are $O(\log \log \log n)$ and $O(n/\log \log \log n)$, respectively. To our knowledge, the previously best known implementation needs $O(\log^2 n)$ time using $O(n/\log n)$ processors on an EREW PRAM. The faster parallel implementation of the tree contraction scheme may be of interests by itself. We then show the above scheme can be utilized to solve problems on distance-hereditary graphs. We provide a data structure to represent a connected distance-hereditary graph in the form of a rooted tree. By applying the above tree contraction scheme on our data structure together with graph theoretical properties, we solve the problems of finding a minimum connected γ-dominating set and finding a minimum γ-dominating clique on a distance-hereditary graph in $O(\log n \cdot T(n))$ time using $O(P(n) + (n + m)/T(n))$ processors on an arbitrary CRCW PRAM, where n and m are the number of vertices and edges of the given graph, respectively. The above result implies several other problems related to the minimum γ-dominating clique problem can be solved with the same parallel complexities.

1. INTRODUCTION

A graph is *distance-hereditary* [2, 25] 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 [14, 21, 25] that are graphs G in which the maximum clique size equals the chromatic number for every induced subgraph of G [3, 20]. Properties of distance-hereditary graphs have been studied by researchers [2, 7, 14, 16, 21, 25] which resulted in sequential algorithms to solve quite a few interesting graph-theoretical problems on this special class of graphs. However, few results [11, 13, 26] are known in the parallel context. In [11], Dahlhaus gave a cost-optimal parallel algorithm to compute the all-to-all vertices distances for a distance-hereditary graph. In [26], efficient parallel algorithms were presented to find a minimum weighted connected dominating set, find a minimum weighted Steiner tree, and find a maximum weighted clique for a given distance-hereditary graph. In this paper, we further study properties of distance-hereditary graphs that will help in designing parallel algorithms in this special class of graphs, which may be of interest by themselves.

Let G be a distance-hereditary graph in which an integer value $\gamma(v)$ is assigned to each vertex v. In this paper, we focus on various generalizations of the γ-dominating set problem, where a γ-dominating set in G is a subset of vertices such that for every vertex $v \in G$ there is a vertex in the γ-dominating set with its distance within $\gamma(v)$. The concept of dominating set is used to model many location problems in operations research and

game theory [6, 8, 24]. We will study the *minimum connected γ-dominating set problem*, i.e., the problem of finding a minimum cardinality γ-dominating set which induces a connected subgraph, and the *minimum γ-dominating clique problem*, i.e., the problem of finding a minimum γ-dominating set which induces a clique. It is easy to see that the minimum connected γ-dominating set problem generalizes the concepts of the well-known minimum connected dominating set problem with $\gamma(v) = 1$ for all vertices and the minimum Steiner tree problem with $\gamma(v) = 0$ for any terminal vertex and $\gamma(v) = \infty$ for other vertices [7, 15]. From solving the γ-dominating clique problem, we show that several related problems can also be solved efficiently in parallel. The sequential linear time algorithms to solve the minimum connected γ-dominating set problem and the minimum γ-dominating clique problem have been presented in [7, 16].

In this paper, we first give an implementation of a parallel tree contraction scheme (described in Section 3) which in each contraction phase removes leaves and nodes in the maximal chains. This scheme was applied to solve several problems on chordal graphs and reducible flow graphs [12, 29, 32–34]. Given an array of n integers $a[1 \ldots n]$, the *all nearest smaller values* (ANSV) problem [4, 5] is for each index i to find the largest index j such that $j < i$ and $a[j] < a[i]$ and to find the smallest index k such that $k > i$ and $a[k] < a[i]$. Let $T(n)$ and $P(n)$ denote the time and processor complexity required to compute the ANSV problem and the minimum of n values for input elements drawn from the integer domain $[1 \ldots n]$. Our implementation takes $O(\log n \cdot T(n))$ time using $P(n)$ processors on an arbitrary CRCW PRAM (concurrent read and write parallel random access machine). Currently, the best results for $T(n)$ and $P(n)$ are $O(\log \log \log n)$ and $O(n/\log \log \log n)$, respectively [5]. To our knowledge, the previously best-known implementation of the above tree contraction scheme needs $O(\log^2 n)$ time using $O(n/\log n)$ processors on an EREW PRAM [29]. We then show the above scheme can help to solve the minimum connected γ-dominating set problem, the minimum γ-dominating clique problem and related problems on distance-hereditary graphs. We provide a data structure to represent a connected distance-hereditary graph in the form of a rooted tree. By applying the indicated tree contraction scheme to prune such a tree, the above problems can be solved in $O(\log n \cdot T(n))$ time using $O(P(n) + (n + m)/T(n))$ processors on an arbitrary CRCW PRAM, where n and m are the number of vertices and edges in the input graph, respectively. The sequential complexity of the above problems is $O(n + m)$ [7, 16].

The computation model used here is the deterministic PRAM which permits CRCW in its shared memory. The arbitrary CRCW PRAM allows an arbitrary processor to succeed [28] when several processors are attempting to write into the same memory location. The rest of this paper is

organized as follows. In Section 2, some needed definitions are given. In Section 3, we provide a new implementation of a tree contraction scheme. In Section 4, a sequential algorithm using our data structure is presented for the minimum connected γ-dominating set problem on a distance-hereditary graph. In Section 5, we present a parallel implementation of our sequential algorithm. In Section 6, the minimum γ-dominating clique problem is discussed. Extensions to several other problems and the conclusion are given in Section 7.

2. PRELIMINARIES

This paper considers a finite, simple, undirected, and connected graph $G = (V, E)$, where V and E are the vertex and edge sets of G, respectively. Let $n = |V|$ and $m = |E|$. For graph-theoretic terminologies and notations not mentioned here, we refer to [20].

Let v be a vertex of G. We denote the number of edges incident to v by $deg_G(v)$ and let $deg(G) = \max\{deg_G(v) | v \in G\}$. We also denote the *neighborhood* of v, consisting of all vertices adjacent to v, by $N_G(v)$, and the *closed neighborhood* of v, the set $N_G(v) \cup \{v\}$, by $N_G[v]$.

Let S be a subset of V. Let $N_G(S)$ denote the *open neighborhood* of S, that is the set of vertices in $V(G) \setminus S$ which are adjacent to any vertex in S. The closed neighborhood of S is the set $N_G(S) \cup S$, which is denoted as $N_G[S]$. The subscript G in the notations can be omitted when no ambiguity arises. The *subgraph induced by* S, denoted by $\langle S \rangle$, is the subgraph with S as the vertex set and $\{(x, y) \in E | x, y \in S\}$ as the edge set. A vertex subset S is *homogeneous* in G if and only if every vertex in $V \setminus S$ is adjacent to either all or none of the vertices of S. A homogeneous set S is further said to be *proper homogeneous* if $2 \le |S| \le n - 1$. Note that every vertex $v \in V \setminus S$ has equal distance to the vertices of a homogeneous set S. We call a family of subsets *arboreal* if every two subsets of the family are either disjoint or comparable (by set inclusion).

For any two vertices u and v, let $dist(u, v)$ denote the distance between u and v in G. Given a vertex $u \in V$, the *hanging* of a connected graph $G = (V, E)$ rooted at u, denoted by h_u, is the collection of sets $L_0(u), L_1(u), \ldots, L_t(u)$ (or simply $L_0, L_1, \ldots, L_t$ when no ambiguity arises), where $t = \max_{v \in V} dist(u, v)$ and $L_i(u) = \{v \in V | dist(u, v) = i\}$ 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 graph G is γ-valued if it is associated with a function γ on V to nonnegative integers. For a γ-valued graph G, a subset $D \subseteq V$ is a

γ-dominating set of G if for every $v \in V \setminus D$, there is a $u \in D$ with $dist(u,v) \leq \gamma(v)$. For a γ-dominating set of G, it is called a *connected γ-dominating set* of G if $\langle D \rangle$ is connected and is called a *γ-dominating clique* of G if D is a clique. The *minimum connected γ-dominating set problem* (respectively, *minimum γ-dominating clique problem*) is the problem of finding a minimum cardinality connected γ-dominating set (respectively, γ-dominating clique) of G.

3. A FASTER IMPLEMENTATION OF A TREE CONTRACTION SCHEME

The problem of *tree contraction* involves reducing in parallel a given tree to its root by a sequence of vertex removals. It has important applications in dynamic expression evaluation and isomorphism testing, among many others [1, 10, 12, 18, 19, 22, 23, 29–34].

We first review a tree contraction scheme used in this paper. The scheme is based upon two abstract parallel tree contraction operations, namely RAKE and SHRINK. The scheme works in phases: during each phase, one RAKE and then one SHRINK operation are performed consecutively.

Let $T = (V, E)$ be a rooted tree with n vertices and $\{v_1, v_2, \ldots, v_k\} \subseteq V$, where $k \geq 2$. We say that $\mathscr{C} = [v_1 v_2, \ldots, v_k]$ is a *chain* of length $k - 1$ if v_1 is not the root, the degree of v_1 is 2, v_{i+1} is the only child of v_i, $1 \leq i < k$, and v_k is a leaf. A chain is said to be *maximal* if it is not possible to add any vertex to form a longer chain. Further, we say that a maximal chain $\mathscr{C} = [v_1, v_2, \ldots, v_k]$ is *reduced* if the vertices $v_2, v_3, \ldots, v_k$ are removed from T. The following two operations are defined in T.

1. SHRINK: An operation reduces all the maximal chains of T. An example of a SHRINK operation is shown in Fig. 1a.

2. RAKE: An operation removes all the leaves from T. An example of a single RAKE operation is shown in Fig. 1b.

We define a *contraction phase* of the current tree by first applying a RAKE operation and then applying a SHRINK operation. The above tree contraction scheme, called $R \& S$ for ease of referencing, applies a sequence of contraction phases to the original tree until it being reduced to its root.

The scheme $R \& S$ was applied to solve several problems on chordal graphs and reducible flow graphs [12, 29, 32–34]. In [33, 34], Ramachandran gave an implementation which needs $O(\log^2 n)$ time using polynomial many processors to solve the minimum feedback vertex set problem on

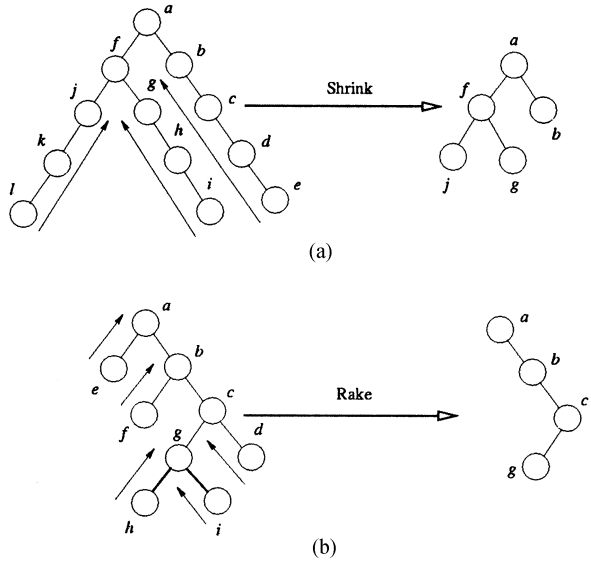


FIG. 1. An example of SHRINK and RAKE operations.

unweighted reducible flow graphs and the minimum feedback arc set on arc-weighted reducible flow graphs. In [32], the scheme was implemented in $O(\log^2 n)$ time using $O(n^2)$ processors to find a perfect elimination order and an unweighted maximum independent set of a chordal graph. In [12], Dahlhaus and Damaschke implemented the scheme in $O(\log^2 n)$ time using $O(n)$ processors on a CREW PRAM based on the pointer jumping technique and used it to solve the dominating set problem and the dominating clique problem on strongly chordal graphs. In [29], Klein implemented the scheme in $O(\log^2 n)$ time using $O(n/\log n)$ processors on an EREW PRAM based on the Euler tour technique and used it to solve the maximum independent set problem on chordal graphs.

LEMMA 1 [12, 29, 32–34]. *After $O(\log n)$ contraction phases, T is reduced to a single vertex which is its root.*

In what follows, we present a method to implement the tree contraction scheme $R \& S$ in $O(\log n \cdot \log\log\log n)$ time using $O(n/\log\log\log n)$ processors on an arbitrary CRCW PRAM. Consider a rooted tree T with root r to be contracted. For a node v in T, let $child_T(v)$ denote the children of v and $par_T(v)$ denote the parent of v in T. Throughput this section, we also use $child(v)$ and $par(v)$ to denote the children and the parent of v in the current tree when no ambiguity occurs. We assume that for each vertex v in T, the children of v are ordered $v_1, v_2, \ldots, v_{l(v)}$,

where $l(v)$ is the number of children of v and each child knows its index. That is, let $index[v_i] = i$ be the index of v_i in this ordering of children. For each vertex v we set aside $l(v)$ locations $label[v, i]$, $i = 1, \ldots, l(v)$ in the shared memory. Initially each $label[v, i]$ is empty or *unmarked*. Once v_j is deleted from the current tree using tree contraction, $label[v, j]$ is marked. We use $arg(v)$ to denote the number of children of v in the current tree. That is, initially, $arg(v) = l(v)$.

Our method works in $O(\log n)$ contraction phases. Assume that the children of the root r are given by $(u_0, u_1, \ldots, u_{l(r)-1})$. First of all, we compute the Euler tour for T which is represented by an array $ET = [e_1 = (r, u_0), e_2, \ldots, e_{2(n-1)} = (u_{l(r)-1}, r)]$ (i.e., $ET[i]$ records the ith edge in the tour constructed). This can be achieved in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM by applying the list ranking technique [27]. For each vertex v, let i_v (respectively, t_v) be the index that $ET[i_v] = (par_T(v), v)$ (respectively, $ET[t_v] = (v, par_T(v))$) and let I_v denote the interval $[i_v, t_v]$. The index i_v (respectively, t_v) is said to be the *left endpoint* (respectively, *right endpoint*) of I_v. For any two intervals I_x and I_y, we say I_x *covers* I_y if $i_x \le i_y$ and $t_y \le t_x$, and they are *disjoint* if $i_x < t_x < i_y < t_y$ or $i_y < t_y < i_x < t_x$.

LEMMA 2. *For any two intervals I_x and I_y, where x, y in $V(T)$, either one covers the other or they are disjoint.*

Proof. According to the property of trees, if x (respectively, y) is an ancestor of y (respectively, x), then I_x covers I_y (respectively, I_y covers I_x); otherwise, I_x and I_y are disjoint. ∎

By Lemma 2, on any subset $\mathcal{I}$ of $\{I_v | v \in V(T)\}$, we can define a partial order $\preceq$ that for any I_x, I_y in $\mathcal{I}$, $I_x \preceq I_y$ if I_y covers I_x.

To indicate the status of vertices in the current tree, we construct an array $\mathcal{D}_{ET}[1, \ldots, 2(n-1)]$ with $2(n-1)$ entries corresponding to the $2(n-1)$ entries of array ET. This array will be updated in each phase as follows. We set $\mathcal{D}_{ET}[i_v] = 1$, or 2, or 3, depending on the degree of v in the current tree is 1 or 2, or at least 3, respectively. We also set $\mathcal{D}_{ET}[j] = 0$ if either $j \neq i_v$ for any $v \in V(T)$ or $j = i_v$ for any $v \in V(T)$ not in the current tree.

In a contraction phase, we first show how to implement RAKE. For each vertex w with $arg(w) = 0$, it is a leaf of the current tree. So we delete all the nodes w with $arg(w) = 0$ (corresponding to a RAKE operation) and modify $\mathcal{D}_{ET}$ as follows. We first set $\mathcal{D}_{ET}[i_w] = 0$. Assume $f = par_T(w)$. We use the following method to compute $\mathcal{D}_{ET}[i_f]$ on $O(1)$ time using totally $O(n)$ processors. Assume each node is assigned with one processor. Using the arbitrary CRCW PRAM model, we start by setting aside a memory location $argindex[f] = null$. Each processor assigned to an unmarked child

writes its index into the memory location *argindex*$[f]$. Assume that one of these children succeeds in writing its index. If *argindex*$[f] = null$, we set $\mathscr{D}_{ET}[i_f] = 1$ because all the children of f are deleted by executing a RAKE operation. To test whether $\mathscr{D}_{ET}[i_f] = 2$, each processor assigned to an unmarked child reads *argindex*$[f]$ and if the value is not the same as its own index, it rewrites its index to *argindex*$[f]$. If the value of *argindex*$[f]$ does not change, then $\mathscr{D}_{ET}[i_f] = 2$; otherwise $\mathscr{D}_{ET}[i_f] = 3$. The above modification can be done in $O(1)$ time using $O(n)$ processors. As with the aid of Brent's scheduling principle [28],[5] we have the following result.

LEMMA 3. *In each contraction phase, a* RAKE *operation can be implemented in* $T^*(n)$ *time using* $P^*(n)$ *processors on an arbitrary CRCW PRAM, for any* $T^*(n)$ *and* $P^*(n)$ *that* $T^*(n) \cdot P^*(n) = O(n)$.

We then show how to implement SHRINK. Let H denote the current tree and $N_{d=\omega} = \{v \in V(H) | deg_H(v) = \omega\}$. To find a maximal chain in the current tree, we first compute the set $Q = \{v \in N_{d=2} | deg_H(par_H(v)) \geq 3$ or $par_H(v)$ is the root of $H\}$ in $O(1)$ time using $O(n)$ processors. We call each vertex $v \in Q$ *chain-leader*. According to the definition of a maximal chain, we have the following lemma.

LEMMA 4. *Let* $\mathscr{I} = \{I_v | v \in Q\}$ *and let* $\mathscr{I}^*$ *denote the set of minimal elements in* $(\mathscr{I}, \preceq)$. *For each* $I_w \in \mathscr{I}^*$, *the subtree rooted at w in the current tree forms a maximal chain if and only if* $max\{\mathscr{D}_{ET}[i] | i_w \leq i \leq t_w\} \leq 2$.

Proof. Straightforward. ∎

Next, we show how to find $\mathscr{I}^*$ in $\mathscr{I}$. For the right endpoint t_v of each I_v in $\mathscr{I}$, we aim at finding an I_u in $\mathscr{I}$ such that $t_u < t_v$ and $t_v - t_u = \min\{t_v - t_w | I_w \in \mathscr{I}, t_w < t_v\}$. If $i_v < i_u$, then $I_v \notin \mathscr{I}^*$. If $i_v > i_u$, then $I_v \in \mathscr{I}^*$. The above problem can be reduced to the ANSV problem as follows. First, we build an array $B[1, \ldots, 2(n-1)]$ corresponding to $2(n-1)$ entries of ET so that each $B[j]$ records $(1, j)$ if $j = t_v$ for some chain-leader v and records $(2, j)$ otherwise. For any two $B[x] = (x_1, x_2)$ and $B[y] = (y_1, y_2)$, we define $B[x] < B[y]$ if either $x_1 < y_1$, or $x_1 = y_1$ and $x_2 < y_2$ holds. By solving the ANSV problem on $B[1, \ldots, 2(n-1)]$, for each $B[t_v] = (1, t_v)$, $v \in Q$, we can find its nearest smaller value $B[j] = (j_1, j_2)$. By definition $j_1 = 1$ and $j_2 = j$. In other words, j is the right endpoint of some I_u in $\mathscr{I}$ which is the closest to t_v with smaller value. If $i_v < i_u$, then $I_v \notin \mathscr{I}^*$. If $i_v > i_u$, then $I_v \in \mathscr{I}^*$. Hence, $\mathscr{I}^*$ can be computed with the same complexity to solve the ANSV problem. For each $I_v \in \mathscr{I}^*$ we find the maximum value t among $\mathscr{D}_{ET}[i_v, \ldots, t_v]$ in $O(\log\log\log n)$ time using

[5] For the rest of this paper, all the implementations which take a constant time using linear number of processors can apply Brent's scheduling principle to achieve the desired complexities.

$O(n/\log\log\log n)$ processors on a common CRCW PRAM [5]. By Lemma 4, the subtree rooted at v in the current tree forms a maximal chain if $t \leq 2$.

After the above computation, we may contract each maximal chain to its chain-leader to complete a contraction phase. At this time, we need to update $\mathscr{D}_{ET}$ with the new degree of each vertex after executing a SHRINK operation. For each chain-leader of a maximal chain w, we set $\mathscr{D}_{ET}[i_w] = 1$ because the degree of w in the current tree is 1 and set each $\mathscr{D}_{ET}[j] = 0$, where $i_v < j < t_v$ because the vertices in the maximal chain are deleted. This takes $O(1)$ time using $O(n)$ processors. After completing a contraction phase and updating $\mathscr{D}_{ET}$, we then go on executing the next phase.

Let $T(n)$ and $P(n)$ denote the time and processor complexity required to compute the ANSV problem and the minimum of n values for input elements drawn from the integer domain $[1 \ldots n]$. The above discussion leads to the following result.

LEMMA 5. *In each contraction phase, a* SHRINK *operation can be implemented in* $O(T(n))$ *time using* $O(P(n))$ *processors on a common CRCW PRAM.*

By Lemmas 3 and 5, we obtain the following theorem.

THEOREM 1. *Algorithm* R & S *can be implemented correctly in* $O(\log n \cdot T(n))$ *time using* $P(n)$ *processors on an arbitrary CRCW PRAM.*

Since the best results for $T(n)$ and $P(n)$ are $O(\log\log\log n)$ and $O(n/\log\log\log n)$, respectively [5], we have the following corollary.

COROLLARY 1. *Algorithm* R & S *can be implemented to run in* $O(\log n \cdot \log\log\log n)$ *time using* $O(n/\log\log\log n)$ *processors on an arbitrary CRCW PRAM, where n is the number of vertices of the input tree.*

4. THE MINIMUM CONNECTED γ-DOMINATING SET PROBLEM

In this section, a sequential algorithm is presented to find a minimum connected γ-dominating set on a distance-hereditary graph. It will be shown in Section 5 that this algorithm can be efficiently parallelized using Algorithm *R & S*. In Section 4.1, we give fundamental results of distance-hereditary graphs. In Section 4.2, we define a data structure, equivalence-hanging tree, to represent a distance-hereditary graph. In Section 4.3, we present a sequential algorithm working on a given equivalence-hanging tree.

4.1. *Previous Known Properties of Distance-Hereditary Graphs*

In the rest of this paper, G denotes a connected distance-hereditary graph whenever no ambiguity occurs.

Fact 1 [2, 14, 21]. Suppose $h_u = (L_0, L_1, \ldots, L_t)$ is a hanging of G rooted at u. If $x, y \in L_i$ $(1 \le i \le t)$ are in the same connected component of $\langle L_i \rangle$ or tied, then $N'(x) = N'(y)$.

Fact 2 [2, 21]. Suppose $h_u = (L_0, L_1, \ldots, L_t)$ is a hanging of G rooted at u. For any two vertices $x, y \in L_i$, $i \ge 1$, $N'(x)$ and $N'(y)$ are either disjoint or one of them is contained in the other.

For a hanging $h_u = (L_0, L_1, \ldots, L_t)$, Hammer and Maffary [21] defined an equivalence relation $\equiv_i$ between vertices of L_i by $x \equiv_i y$ 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 $x \equiv_a y$ means $x \equiv_i y$ for some i. The equivalence relation $\equiv_a$ partitions $V(G)$ into equivalence classes. For an equivalence class R, let $\Gamma_R = \{R\} \cup \{S \subseteq R |$ there is an equivalence class R' with $N'(R') = S\}$. Γ_R is called the *upper neighborhood system in* R and S $(= N'(R'))$ in Γ_R the *upper neighborhood* of R'.

Fact 3 [21]. Let h_u be the hanging of G rooted at u and R be any equivalence class with respect to h_u. Γ_R is an arboreal family of homogeneous subsets of $\langle R \rangle$.

For any equivalence class R, we define a partial order $\preceq$ between two different sets S_p and S_q in Γ_R by $S_p \preceq S_q \Leftrightarrow S_p \subset S_q$. S_q is called to *immediately succeed* S_p if $S_p \preceq S_q$ and there is no $S_r \in \Gamma_R$ such that $S_p \preceq S_r \preceq S_q$.

4.2. *The Equivalence-Hanging Tree*

Let $\mathscr{E} = \{R_1, R_2, \ldots, R_k\}$ be the set of equivalence classes of G with respect to h_u. We define a graph $T_{h_u} = (V(T_{h_u}), E(T_{h_u}))$, as follows. For each S in $\bigcup_{R \in \mathscr{E}} \Gamma_R$, we create a node for T_{h_u} to represent S. There are totally $\sum_{i=1}^{k} |\Gamma_{R_i}|$ created nodes. For each node $\omega \in V(T_{h_u})$, let S_ω denote the vertex set represented by ω. For $\alpha, \beta \in V(T_{h_u})$, (α, β) is an edge of T_{h_u} if it satisfies one of the following two conditions: (1) $S_\alpha, S_\beta \in \Gamma_R$ for some $R \in \mathscr{E}$ and S_β immediately succeeds S_α in Γ_R; (2) S_α is an equivalence class and $S_\beta = N'(S_\alpha)$. An edge satisfies condition (1) (respectively, condition (2)) is called an *abnormal edge* (respectively, a *normal edge*). Let ρ be the node in T_{h_u} that $S_\rho = \{u\}$ (u is the root of the given hanging). By the above definition, there exists no node $\nu \in V(T_{h_u})$ such that $(\rho, \nu) \in E(T_{h_u})$, and for any $\alpha \ne \rho$, there exists a unique μ such that $(\alpha, \mu) \in E(T_{h_u})$. Therefore, there are exactly $|V(T_{h_u})| - 1$ edges in T_{h_u}.

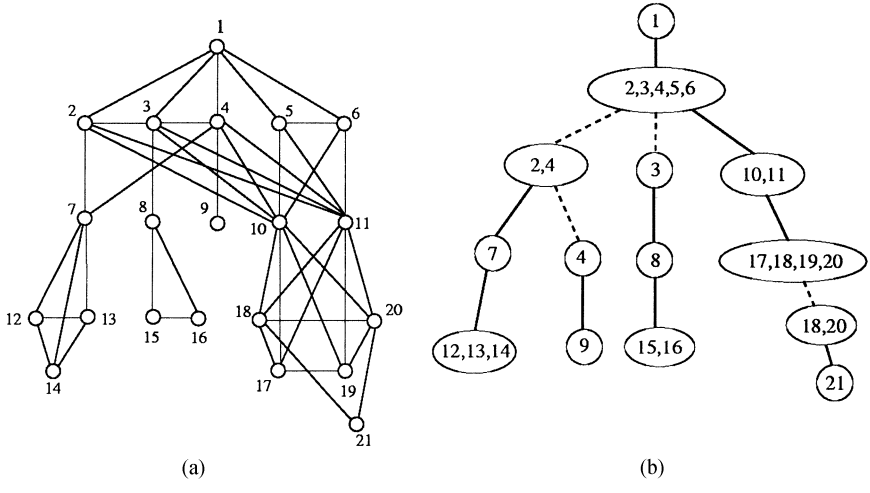
LEMMA 6. *The graph T_{h_u} is a tree and $|V(T_{h_u})| = O(n)$.*

Proof. Since there are exactly $|V(T_{h_u})| - 1$ edges in T_{h_u}, to show T_{h_u} is a tree, it suffices to show it contains no cycle. Suppose, by contrast, that there is a cycle $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ in T_{h_u}. Let $S_{\alpha_i} \subseteq L_{q_i}$. It leads to $q_1 \geq q_2 \geq \cdots \geq q_k \geq q_1$. Thus, $q_1 = q_2 = \cdots = q_k$. This implies that $S_{\alpha_1} \preceq S_{\alpha_2} \preceq \cdots \preceq S_{\alpha_k} \preceq S_{\alpha_1}$, a contradiction. Thus, T_{h_u} is a tree.

Since there are at most n equivalence classes, $|\Gamma_{R_1}| + |\Gamma_{R_2}| + \cdots + |\Gamma_{R_k}| = O(n)$. Therefore, $|V(T_{h_u})| = O(n)$. ∎

We call T_{h_u} the *equivalence-hanging tree* of the given distance-hereditary graph G with respect to h_u. For the rest of this paper, we assume T_{h_u} is a tree rooted at the node representing $\{u\}$, which is an equivalence class. For a node ν in the rooted tree T_{h_u}, we denote $Nchild(\nu) = \{\beta \in child_{T_{h_u}}(\nu) | (\beta, \nu)$ is a normal edge$\}$ and $Achild(\nu) = \{\beta \in child_{T_{h_u}}(\nu) | (\beta, \nu)$ is an abnormal edge$\}$. Figure 2b shows an equivalence-hanging tree with respect to the hanging at vertex 1 of the distance-hereditary graph illustrated in Fig. 2a.

In the following, we consider a process that reduces T_{h_u} to its root node. In an iteration of the reduction process, a leaf node of the current tree is removed. Let $T^i_{h_u}$ denote the resulted tree after the ith iteration and G^i denote the subgraph of G induced by $\bigcup_{\nu \in V(T^i_{h_u})} S_\nu$, for $i \leq |V(T_{h_u})|$. Note that by the hereditary property of distance-hereditary graphs, G^i is also a



(a) (b)

FIG. 2. An equivalence-hanging tree shown in (b) of a distance-hereditary graph shown in (a). The bold lines show the normal edges and the dashed lines show the abnormal edges. The label (a sequence of numbers) of each node ν in the equivalence-hanging tree represents S_ν.

distance-hereditary graph. For G^i we may consider its hanging rooted at u (u is the last removed vertex), denoted by h_u^i.

LEMMA 7. *For every* $i \le |V(T_{h_u})|$, G^i *is connected and* $h_u^i = (L_0 \cap V(G^i), \ldots, L_t \cap V(G^i))$.

Proof. We prove the lemma by induction on i. When $i = 0$, $G^0 = G$ is connected and $h_u^0 = h_u$. Suppose that G^{i-1} is connected and $h_u^{i-1} = (L_0 \cap V(G^{i-1}), \ldots, L_t \cap V(G^{i-1}))$. Assume that $T_{h_u}^i$ is obtained by removing the leaf node ν in $T_{h_u}^{i-1}$. Let the parent of ν be β. If S_ν is not in $\mathscr{E}$, since $S_\nu \subset S_\beta$, $G^i = G^{i-1}$; thus, G^i is connected and $h_u^i = (L_0 \cap V(G^i), \ldots, L_t \cap V(G^i))$. Suppose that S_ν is in $\mathscr{E}$ and $S_\nu \subset L_j$. Let v be any vertex in $V(G^{i-1}) \setminus S_\nu$ and $v \in L_k \cap V(G^{i-1})$. It suffices to show that v is of distance k to u in G^i. Moreover, we show that any shortest path from v to u in G^{i-1} is also a shortest path in G^i. Let $P = [v_k(= v), v_{k-1}, \ldots, v_0(= u)]$ be any shortest path from v to u. By definition, $v_s \in L_s \cap V(G^{i-1})$ for $0 \le s \le k$. Suppose the contrary that P contains a vertex w in S_ν. That is, $k > j$ and $w = v_j$. It implies that v_{j+1} in $L_{j+1} \cap V(G^{i-1})$ and $(v_{j+1}, w) \in E(G^i)$. However, v_{j+1} has to be in S_ρ where (ρ, ν) is a normal edge. It contradicts that ν is a leaf node in $T_{h_u}^{i-1}$. Hence, G^i is connected and $h_u^i = (L_0 \cap V(G^i), \ldots, L_t \cap V(G^i))$. ∎

For any v in $V(G^i)$, $v \in S_\alpha$ for some $\alpha \in V(T_{h_u}^i)$. Since $par(\alpha)$ is also in $V(T_{h_u}^i)$, $N'_G(v) \subset V(G^i)$. Let $N'_{G^i}(v)$ denote the upper neighborhood of v in $V(G^i)$ with respect to h_u^i. By Lemma 7, for any vertex v in $V(G^i)$, we have $N'_{G^i}(v) = N'_G(v)$.

For a rooted tree T, let $leaf(T)$ denote the leaves of T and for $v \in V(T)$ let $T(v)$ denote the subtree of T with root v.

LEMMA 8. *If* $\nu \in leaf(T_{h_u}^i)$, *then* S_ν *is a homogeneous set of* G^i.

Proof. By definition, either $S_\nu \in \mathscr{E}$ or $S_\nu = N'(R)$ for some $R \in \mathscr{E}$. First, suppose that S_ν is in $\mathscr{E}$. By Fact 1, for arbitrary two vertices $x, y \in S_\nu$, $N'_G(x) = N'_G(y)$ and thus $N'_{G^i}(x) = N'_{G^i}(y)$. Since $\nu \in leaf(T_{h_u}^i)$, $N'_{G^i}(S_\nu) = N_{G^i}(S_\nu)$. By definition, S_ν is a homogeneous set of G^i. Next, suppose that S_ν is the upper neighborhood of some equivalence class of G. Assume that S_ν is a proper subset of a set Q in $\mathscr{E}$. Let $\mathscr{C} = \{\beta | N'(S_\beta) \subseteq S_\nu\}$. We have that $(\bigcup_{\beta \in \mathscr{C}} S_\beta) \cap V(G^i) = \varnothing$. Moreover, S_ν is a homogeneous set in $\langle Q \rangle$ by Fact 3, and $N'_{G^i}(x) = N'_{G^i}(y)$ for any two vertices x and y in S_ν. Thus, S_ν is a homogeneous set of G^i. ∎

LEMMA 9. *Suppose* ν *is a node of* $T_{h_u}^i$. *Let* $A = Achild_{T_{h_u}}(\nu) \cap V(G^i)$. *If* $A \subseteq leaf(T_{h_u}^i)$, *then* S_ν *is a homogeneous set of* G^i.

Proof. Similar to the proof of Lemma 8. ∎

We next describe a method to construct T_{h_u}. We first compute the equivalence classes with respect to h_u in $O(\log n)$ time using $O(n + m)$ processors on an arbitrary CRCW PRAM [26]. Using Cole's parallel merge sorting [9], the upper neighborhoods of the equivalence classes can be computed in $O(\log n)$ time using $O(n + m)$ processors on an EREW PRAM [26]. Thus, $V(T_{h_u})$ can be created with the desired complexity. For each upper neighborhood X in an upper neighborhood system Γ_R, a method shown in [26] can be used to find the upper neighborhood $Y \in \Gamma_R$ that immediately succeeds X in Γ_R. This takes $O(\log n)$ time using $O(n + m)$ processors on an arbitrary CRCW PRAM [26]. According to the above computation, the parent of each node in $V(T_{h_u})$ can be found in constant time. It is easy to determine the type of each edge in $E(T_{h_u})$. Applying Brent's scheduling principle, the following result is obtained.

LEMMA 10. *Given a distance-hereditary graph G, T_{h_u} can be constructed in $O(\log n \cdot \log\log\log n)$ time using $O((n + m)/\log\log\log n)$ processors on an arbitrary CRCW PRAM.*

4.3. *A Sequential Algorithm*

A linear time sequential algorithm is first described in [7]. Here we present another sequential algorithm used in the next section for parallelization. Let G' be an induced subgraph of G with a new γ value γ' assigned to each of its vertices. Let $D(G)$ and $D(G')$ denote a minimum connected γ-dominating set of G and a minimum connected γ'-dominating set of G', respectively. Fact 4 shows that by properly choosing G' and setting γ' values, we can reduce the problem of computing $D(G)$ to the problem of computing $D(G')$.

For a vertex x in a homogeneous set $\mathscr{C}$ of G, let $tag(x) = 1$ if there is a vertex $y \in V(G)\backslash\mathscr{C}$ with $dist(x, y) > \gamma(y)$, and let $tag(x) = 0$ for otherwise.

Fact 4 [7]. Let $\mathscr{C}$ be a vertex subset of the given graph $G = (V, E)$.

(a) Assume that $\mathscr{C} \subset V$ is a proper homogeneous set of G. Let x be a vertex of $\mathscr{C}$ with $\gamma(x) = \min\{\gamma(y) | y \in \mathscr{C}\}$. Also let $G' = \langle (V \backslash \mathscr{C}) \cup \{x\} \rangle$ and $\gamma'(v) = \gamma(v)$ for all $v \in G'$. (i) If $\gamma(x) \geq 2$, or $\gamma(x) = 1$ and $tag(x) = 1$, then $D(G) = D(G')$. (ii) If $\gamma(x) = 0$ and $\gamma(w) = 0$ for some vertex $w \in V \backslash \mathscr{C}$, then $D(G) = D(G') \cup \{y \in \mathscr{C} | \gamma(y) = 0\}$.

(b) Assume that $\mathscr{C}$ contains only one vertex x such that $N(x)$ forms a homogeneous set in G. Let y be a vertex of $N(x)$ with $\gamma(y) = \min\{\gamma(z) | z \in N(x)\}$. Also let $G' = \langle V \backslash \{x\} \rangle$ and $\gamma'(v) = \gamma(v)$ for all $v (\neq y) \in G$. (i) If $\gamma(x) \geq 2$, or $\gamma(x) = 1$ and $tag(x) = 1$, then $D(G) = D(G')$ with $\gamma'(y) = \min\{\gamma(y), \gamma(x) - 1\}$. (ii) If $\gamma(x) = 0$ and $\gamma(w) = 0$ for some vertex $w \in V \backslash N[x]$, then $D(G) = D(G') \cup \{x\}$ with $\gamma'(y) = 0$.

Fact 5 [7]. Suppose x is a vertex of a homogeneous set $\mathscr{C}$ with $\gamma(x) = \min\{\gamma(y) | y \in \mathscr{C}\} = 1$ and $tag(x) = 0$. If there is a vertex $w \in \mathscr{C}$ satisfying $N[w] \supseteq \{v \in \mathscr{C} | \gamma(v) = 1\}$, then $D(G) = \{w\}$; otherwise, $D(G) = \{x, z\}$, where $z \in N(\mathscr{C})$.

For a given γ-valued distance-hereditary graph G, the sequential algorithm in [7] processes as follows. If there is a vertex $u \in V(G)$ whose γ-value is 0, we compute h_u; otherwise, we compute a hanging rooted at an arbitrary vertex u. Let $h_u = (L_0, L_1, \ldots, L_t)$. The sequential algorithm in [7] processes G from the bottom layer L_t to the top one L_0 based on Facts 4 and 5. When L_i is processed, G is reduced to the induced subgraph $\langle \bigcup_{j=0}^{i} L_j \rangle$. Then, the algorithm finds the connected components of $\langle L_i \rangle$. Each connected component is a homogeneous set of the graph $\langle \bigcup_{j=0}^{i} L_j \rangle$. The reason is explained below. Let H be a connected component of $\langle L_i \rangle$. According to Fact 1, $N'(x) = N'(y) = N'(H)$ for every two vertices x and y in H. Therefore, every vertex of $N'(H)$ is adjacent to every vertex of H. By definition, H is a homogeneous set of the graph $\langle \bigcup_{j=0}^{i} L_j \rangle$. Next, the connected components are ordered increasingly according to the cardinalities of their upper neighborhoods. Then the algorithm removes components from G one at a time starting from the one with the smallest order. According to Facts 4 and 5, the new γ-value is adjusted for the resulting graph after each removal. If the γ-value of the root of the hanging is not 0, once the γ-value of some vertex v is adjusted to 0, the algorithm computes a hanging rooted at v of the current graph and then continues the process on the new hanging from the bottom layer of the hanging to the top one as above. The information of a minimum connected γ-dominating set of G is gathered from the adjusted γ-values.

By Facts 4 and 5, to compute the minimum connected γ-dominating set of G, we may choose any homogeneous set of current graph to reduce in each iteration of the algorithm. This property facilitates obtaining an efficient parallel algorithm for the minimum connected γ-dominating set problem. By Lemma 8, the sets represented by the leaves of the equivalence-hanging tree are homogeneous, which can be reduced. For better understanding of the parallel algorithm implementation, we first describe a sequential algorithm using the reduction order induced by removing nodes of the equivalence-hanging tree from leaves to the root.

Let $\nu_1, \nu_2, \ldots, \nu_k$ be any order of vertices of T_{h_u} such that ν_i is a leaf of the induced subtree $T_{h_u}^{i-1} = \langle \nu_i, \ldots, \nu_k \rangle$, for all $i = 1, 2, \ldots, k$. Let G^i be the subgraph induced by $\bigcup_{j=i+1}^{k} S_{\nu_j}$. By Lemma 8, S_{ν_i} is a homogeneous set of G^{i-1}. Thus, processing $S_{\nu_1}, \ldots, S_{\nu_k}$ in order, we can also obtain an algorithm for the minimum connected γ-dominating set problem. In the ith iteration, the algorithm removes node ν_i in $T_{h_u}^{i-1}$ as well as S_{ν_i} if it is

an equivalence class and properly updates the γ-values according to Facts 4 and 5. For clarity of algorithm presentation, in the following, for each v in $V(G)$, we denote its resulted γ-value of the ith iteration by $\gamma_i(v)$. We next describe the additional data structures, $\gamma(\alpha)$, $\delta(\alpha)$, and Λ_α associated with node α in the given equivalence-hanging tree, where the former two variables are integers and the latter one is a set of vertices of $V(G)$. These three variables associated with α are updated whenever a node in $child_{T_{h_u}}(\alpha)$ is removed in the execution of the algorithm. We also denote their resulted values of the ith iteration by $\gamma_i(\alpha)$, $\delta_i(\alpha)$, and $\Lambda_{\alpha,i}$, respectively. For a node α in $V(T_{h_u})$, let $\tilde{S}_{\alpha,i} = S_\alpha \setminus \bigcup_{\nu \in Achild_{T^i_{h_u}}(\alpha)} S_\nu$ if $\alpha \in V(T^i_{h_u})$ and $\tilde{S}_{\alpha,i} = S_\alpha$ otherwise. Initially, for each $\alpha \in V(T_{h_u})$, $\gamma(\alpha) = \gamma_0(\alpha) = \min\{\gamma_0(v) | v \in \tilde{S}_{\alpha,0}\}$. For each $\alpha \in V(T_{h_u})$, $\delta_i(\alpha)$ is defined to be $\min\{\gamma_i(v) | v \in \tilde{S}_{\alpha,i}\}$ and $\Lambda_{\alpha,i}$ is defined to be $\{v \in \tilde{S}_{\alpha,i} | \gamma_i(v) = \delta_i(\alpha)\}$ for all i. Note that initially $\gamma(\alpha)$ equals $\delta(\alpha)$ for all α, but they are not always equal in the execution of the algorithm.

We now present a high level description of our sequential algorithm, called SCD, to find a minimum connected γ-dominating set $D(G)$. If there is a vertex $u \in V(G)$ whose γ-value is 0, we compute the hanging h_u; otherwise, we compute a hanging root at an arbitrary vertex, u. Next, an equivalence-hanging tree T_{h_u} is constructed. Assume that we are in the $(i+1)$th iteration. Let α be the leaf to be removed in $T^i_{h_u}$ which is not the root. Also let $par(\alpha) = \beta$. Note that S_α is a homogeneous set of graph G^i by Lemma 8. We process α as follows.

Case 1. $\gamma_i(u) > 0$, where u is the root of the current hanging.

Case 1.1 $\gamma_i(\alpha) = 0$. In this case, there exists a node ω in $Nchild_{T^0_{h_u}}(\alpha)$ with $\gamma_i(\omega) = 1$. /* *ω is not in $T^i_{h_u}$.* */ Pick an arbitrary vertex x in $\Lambda_{\omega,i}$. Let $tag(x) = 1$ if there is a vertex $v \in v(G^i)$ with $dist(x, v) > \gamma_i(v)$; otherwise, let $tag(x) = 0$.

Case 1.1.1 $tag(x) = 1$. Pick a vertex $y \in \Lambda_{\alpha,i}$. /* Note that $\gamma_i(y) = 0$. */ Let G' denote the subgraph of G induced by $(\bigcup_{\nu \in V(T^0_{T_{h_u}}) \setminus V(T^0_{T_{h_u}}(\alpha))} S_\nu) \cup S_\alpha$. Let $\gamma_{i+1}(v) = \gamma_0(v)$ for all $v \in V(G') \setminus S_\alpha$. Determine the handing h_y of G'. Construct the equivalence-hanging tree T_{h_y}. Replace $T^i_{h_u}$ with T_{h_y}, and go to the next iteration.

Case 1.1.2. $tag(x) = 0$. /* *x γ-dominates all vertices of $V(G) \setminus S_\omega$.* */ If there is a vertex $w \in S_\omega$ satisfying $N[w] \supseteq \Lambda_{\omega,i}$, output $D(G) = \{w\}$ and terminate the execution; otherwise, output $D(G) = \{x, z\}$, where z is an arbitrary vertex in $N'(S_\omega)$, and terminate the execution. /* *based on Fact 5.* */

Case 1.2. $\gamma_i(\alpha) \geq 1$.

Case 1.2.1. (α, β) is a normal edge. Let $\gamma_{i+1}(\beta) = \min\{\gamma_i(\alpha) - 1, \gamma_i(\beta)\}$. Remove α and S_α from $T^i_{h_u}$ and $V(G^i)$, respectively. If α is the only child of β in $T^i_{h_u}$ and $\gamma_{i+1}(\beta) < \delta_i(\beta)$, then pick an arbitrary vertex g in $\Lambda_{\beta,i}$, let $\gamma_{i+1}(g) = \gamma_{i+1}(\beta)$, and then $\Lambda_{\beta,i+1} = \{g\}$.

Case 1.2.2. (α, β) is an abnormal edge. Let $\gamma_{i+1}(\beta) = \min\{\gamma_i(\alpha), \gamma_i(\beta)\}$. If $\gamma_i(\alpha) = \delta_i(\beta)$, then $\Lambda_{\beta,i+1} = \Lambda_{\alpha,i} \cup \Lambda_{\beta,i}$. If $\gamma_i(\alpha) < \delta_i(\beta)$, then let $\delta_{i+1}(\beta) = \delta_i(\alpha)$ and $\Lambda_{\beta,i+1} = \Lambda_{\alpha,i}$. Remove α from $T^i_{h_u}$. If α is the only child of β in $T^i_{h_u}$ and $\gamma_{i+1}(\beta) < \delta_{i+1}(\beta)$, then pick an arbitrary vertex g in $\Lambda_{\beta,i+1}$, let $\gamma_{i+1}(g) = \gamma_{i+1}(\beta)$, and $\Lambda_{\beta,i+1} = \{g\}$.

Case 2. $\gamma_i(u) = 0$. If (α, β) is a normal edge, let $\gamma_{i+1}(\beta) = \max\{0, \min\{\gamma_i(\alpha) - 1, \gamma_i(\beta)\}\}$; otherwise, let $\gamma_{i+1}(\beta) = \max\{0, \min\{\gamma_i(\alpha), \gamma_i(\beta)\}\}$, using the same method as Case 1.2 to maintain $\delta_{i+1}(\beta)$, $\Lambda_{\beta,i+1}$, the current tree, and the current graph.

Algorithm SCD works by repeatedly executing the above two cases until either $D(G)$ is found or T_{h_u} is reduced to its root, where T_{h_u} is the equivalence-hanging tree of the rehanging if a rehanging occurs. If no rehanging occurs and no vertex whose γ-value becomes 0 during the execution, then $D(G)$ is the root of the given hanging; otherwise, $D(G) = \{w|\gamma(w) = 0\}$.

Note that only if Case 1.1.1 is performed, a rehanging occurs. When a rehanging occurs, the γ-value of the root of the rehanging is 0; then only Case 2 is performed in the following iterations and no more rehanging occurs.

Before showing the correctness of the algorithm, we define some notations. If a rehanging occurs, the iterations before (respectively, after) the rehanging are called *valid* for the nodes in the equivalence-hanging tree of the initial hanging (respectively, the rehanging). For a given equivalence-hanging tree T_{h_u}, we say a node β is in a *complete state* when nodes in $child_{T_{h_u}}(\beta)$ are all deleted and β is in an *incomplete state* if some but not all of its children are deleted. The *critical value* $cri(\beta)$ for all $\beta \in V(T_{h_u})$ is defined recursively as follows. If $\beta \in leaf(T_{h_u})$, $cri(\beta) = \min\{\gamma_0(v)|v \in S_\beta\}$. For an arbitrary node β not in $leaf(T_{h_u})$, let $cri(\beta) = \min\{q, r, s - 1\}$ if $\gamma_0(u) \neq 0$ and $cri(\beta) = \max\{0, \min\{q, r, s - 1\}\}$ if $\gamma_0(u) = 0$, where $q = \min\{\gamma_0(v)|v \in \tilde{S}_{\beta,0}\}$, $r = \min\{cri(\nu)|\nu \in Achild_{T^0_{h_u}}(\beta)\}$, and $s = \min\{cri(\nu)|\nu \in Nchild_{T^0_{h_u}}(\beta)\}$. Note that in the case of $\gamma(u) \neq 0$, these nodes with negative critical values will not be processed in the algorithm, since according to Case 1.1, the minimum connected γ-dominating set is found or a new hanging is created.

Suppose that following a reduction order, a rehanging occurs in the ith iteration when processing node α and y is the root of the hanging of G' as in Case 1.1.1. Note that since $\gamma_{i+1}(v)$ is reset to $\gamma_0(v)$ for all $v \in V(G') \setminus S_\alpha$, Algorithm SCD following the reduction order has done up to the end of the ith iteration is equivalent to that following a reduction order $\alpha_1, \ldots, \alpha_k = \alpha$, where $\{\alpha_j\}_{j=1}^{k}$ is the set of nodes in the subtree rooted at α. In other words, in this case, we may assume that Algorithm SCD reduces the subtree rooted at α of the initial equivalence-hanging tree T_{h_u} to the node α, creates the hanging tree h_y of G', and then reduces the equivalence-hanging tree of the rehanging T_{h_y} to its root. With this assumption, no node is in an incomplete state when the rehanging occurs.

LEMMA 11. *Let $\beta \in V(T_{h_u})$, where h_u is the initial hanging or the rehanging, and let k be the first iteration that β becomes a leaf in T_{h_u}. Then $\gamma_j(\beta) = cri(\beta)$ for all valid iterations $j \geq k$ for β.*

Proof. We will prove the case of the initial hanging. The proof for the case of the rehanging is the same. Since after β becomes a leaf, the γ-value of β will not change any more, it suffices to prove that $\gamma_k(\beta) = cri(\beta)$. We prove the lemma by induction on the height of β in T_{h_u}. If β is of height 0, it is a leaf of T_{h_u} and thus $k = 0$. By definition, $cri(\beta) = \gamma_0(\beta)$. Suppose for all nodes of height less than h, the lemma is true. Let β be a node of height h and let α be any child of β in T_{h_u}. Let i_α be the first iteration that α becomes a leaf of $T_{h_u}^{i_\alpha}$. Thus, $i_\alpha < k$ for all $\alpha \in child_{T_{h_u}}(\beta)$. Since the height of α is $h - 1$, by induction, $cri(\alpha) = \gamma_j(\alpha)$ for all $j \geq i_\alpha$. Now we assume the kth iteration is valid for β. Then the lemma is implied by the γ-value updating rules in Cases 1.2 and 2 in the kth iteration, in which $\gamma_{k-1}(\alpha)$ is actually $cri(\alpha)$. ∎

As a node α becomes a leaf, the set of vertices in S_α with γ-values equal to $cri(\alpha)$ is called the *critical vertex set* of α. By Lemma 11, the critical vertex set of α is exactly $\Lambda_{\alpha, i_\alpha}$.

THEOREM 2. *Algorithm* SCD *correctly finds a minimum connected γ-dominating set for a distance-hereditary graph G in $O(n + m)$ time and space.*

Proof. If a rehanging occurs, as discussed in the paragraph before Lemma 11, we make an assumption on the reduction order of Algorithm SCD that only the nodes in the subtree rooted at α are reduced before the rehanging, where α is the node processed in Case 1.1.1.

First, we prove that Algorithm SCD is correct if it follows a particular reduction order. It suffices to show that the reduction in each iteration satisfies Facts 4 and 5 [7]. A reduction order of T_{h_u} is *special* if for any node β in $V(T_{h_u})$, any child in $Nchild_{T_{h_u}}(\beta)$ is removed after all the children in $Achild_{T_{h_u}}(\beta)$ are removed. It is easy to see that there exists a

special reduction order that satisfies the assumption on the reduction order of Algorithm SCD. Following a special reduction order, whenever a node α with $S_\alpha \in \mathscr{E}$ is removed in an iteration, by Lemmas 8 and 9, both S_α and $S_{par(\alpha)}$ are homogeneous sets in the iteration. The reduction in Cases 1.2.1 and 2 can be looked as shrinking S_α to a vertex first as in Fact 4(a) and then removing the vertex as in Fact 4(b). In Case 1.2.1, the γ-value passed from S_α is recorded in $\gamma(par(\alpha))$, and the vertex in $S_{par(\alpha)}$ to record the passed value is maintained by $\Lambda_{par(\alpha)}$. Since S_α and $S_{par(\alpha)}$ are homogeneous sets, the reduction satisfies Fact 4. Case 1.1.2 satisfies Fact 5. As Case 1.1.1 is performed, a new hanging is created and in Case 1.2.2, the vertex of minimum γ-value in $S_{par(\alpha)}$ is maintained. No reduction is performed in Cases 1.1.1 and 1.2.2. Thus, executing Algorithm SCD following a special reduction order is correct.

Now consider an arbitrary reduction order that satisfies the assumption on reduction order of Algorithm SCD. In any iteration a node β is processed, β is in a complete state (it is a leaf of current equivalence-hanging tree). By Lemma 11, the γ-value of β is the same as that computed by the algorithm following a special reduction order. In the meantime, Λ_β contains a vertex of minimum γ-value in S_β. In other words, when reducing β, the update of γ-values in S_β is the same as that in the algorithm following a special reduction order. Therefore, Algorithm SCD following arbitrary reduction order is correct.

The following reasons assert the time complexity of the algorithm. Without loss of generality, we assume the γ-value of the root of a given hanging is nonzero. When the algorithm processes a leaf node α with $\gamma(\alpha) = 0$, $tag(x)$ is first determined in linear time. If $tag(x) = 1$, then a new equivalence-hanging tree is constructed in linear time. The following iterations aim at finding those vertices with γ-value zero after each reduction, and the reconstruction of an equivalence-hanging tree cannot occur. If $tag(x) = 0$, then a minimum connected γ-dominating set is generated and the algorithm is terminated. Note that the tag value is computed at most once in the whole execution. For the other values of $\gamma(\alpha)$, the time to process α is $O(1)$. Therefore, the algorithm runs in linear time and space. ▌

5. FINDING A MINIMUM CONNECTED γ-DOMINATING SET IN PARALLEL

In this section, we show that the utilization of the equivalence-hanging tree and the tree contraction scheme $R\&S$ make the parallelism of Algorithm SCD possible. Given a distance-hereditary graph G in the form

of its equivalence-hanging tree T_{h_u}, recall that Algorithm SCD removes leaves of T_{h_u} one at a time. After removing a node v, the algorithm either outputs a minimum connected γ-dominating set or updates the γ-values associated with some nodes in current tree and graph. Though the above process seems to be highly sequential, we observe that some proper subset $Q \subset V(T_{h_u})$ can be removed simultaneously without affecting the computation of the γ-values associated with $V(T_{h_u})$. In other words, the adjustment of γ-values caused by one node in Q does not affect the adjustment caused by each of the other nodes of Q. We further observe that the nodes removed by a SHRINK or RAKE operation satisfy the above property.

Let $v \in leaf(T^i_{h_u})$ and $child_{T^0_{h_u}}(v) = \{\mu_1, \mu_2, \ldots, \mu_k\}$. In executing Algorithm SCD, the situation that $\gamma(v)$ equals $cri(v)$ occurs after deleting $child_{T^0_{h_u}}(v)$. In deleting each μ_j, we provide $v(\mu_j) = cri(\mu_j)$ as an *input value* to update $\gamma(v)$ depending on the type of (μ_j, v). Recall that when nodes in $child_{T^0_{h_u}}(v)$ are all deleted, we say v is in a *complete state*. Moreover, we say v is in an *almost complete state without* μ if $cri(v)$ can be computed by giving $cri(\mu)$.

5.1. *Algorithms for RAKE and SHRINK*

We first briefly describe how our parallel algorithm works as follows. If there is a vertex $u \in V(G)$ with $\gamma(u) = 0$, we construct T_{h_u}; otherwise, we construct an equivalence-hanging tree T_{h_u}, where u is an arbitrary vertex. The initial values and the data structure used to maintain T_{h_u} in the computation are the same as the ones used in Algorithm SCD. We then design algorithms executed with RAKE and SHRINK to adjust γ-values of the current tree and graph such that $D(G)$ can be generated consequently using Algorithm $R \& S$.

5.1.1. *Algorithms for RAKE.* Suppose $W = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ is a maximal set of leaves in $T^i_{h_u}$ which have the common parent, denoted by $par(W) = par(\alpha_j) = \beta$. We will refer W by a *maximal common-parent leaf set* for convenience. Let $r = \min\{\gamma_i(\alpha_j) | \alpha_j \in Achild_{T^i_{h_u}}(\beta) \cap W\}$ and $s = \min\{\gamma_i(\alpha_j) | \alpha_j \in Nchild_{T^i_{h_u}}(\beta) \cap W\}$. Below are two algorithms applied with RAKE on W.

ALGORITHM R1. /* works on W when $\gamma_i(u) > 0$. */

Case 1. $\min\{r, s\} = 0$. Find a node $\alpha \in W$ such that $\gamma_i(\alpha) = 0$. Let ω be a node in $child_{T^0_{h_u}}(\alpha)$ satisfying $\gamma_i(\omega) = cri(\omega) = 1$ and (ω, α) is normal. Pick an arbitrary vertex x in $\Lambda_{\omega, i}$. Determine $tag(x)$ as in Case 1.1 of Algorithm SCD. If $tag(x) = 1$, then execute Case 1.1.1 of Algorithm SCD; otherwise, execute Case 1.1.2 of Algorithm SCD.

Case 2. $\min\{r, s\} \geq 1$. Let $\gamma_{i+1}(\beta) = \min\{\gamma_i(\beta), r, s-1\}$. For each $\alpha \in W$, remove α from $T^i_{h_u}$, and for each $\alpha \in Nchild_{T^i_{h_u}}(\beta) \cap W$, remove S_α from $V(G^i)$. Let $l = \min\{\delta_i(\beta), r\}$ and $\mathscr{P} = \{\Lambda_{\nu,i} | \nu \in Achild_{T^i_{h_u}}(\beta) \cap W$ and $\gamma_i(\nu) = l\}$. If $l < \delta_i(\beta)$, then let $\Lambda = \bigcup_{X \in \mathscr{P}} X$; otherwise let $\Lambda = (\bigcup_{X \in \mathscr{P}} X) \cup \Lambda_{\beta,i}$;

Case 2.2.1. $W = child_{T^i_{h_u}}(\beta)$ and $\gamma_{i+1}(\beta) < l$. Select an arbitrary vertex x from Λ. Let $\gamma_{i+1}(x) = \gamma_{i+1}(\beta)$ and $\Lambda_{\beta,i+1} = \{x\}$.

Case 2.2.2 $W \subset child_{T^i_{h_u}}(\beta)$ or $\gamma_{i+1}(\beta) \geq l$. Let $\delta_{i+1}(\beta) = l$ and $\Lambda_{\beta,i+1} = \Lambda$.

ALGORITHM R2. /* works on W when $\gamma_i(u) = 0$. */

Let $\gamma_{i+1}(\beta) = \max\{0, \min\{\gamma_i(\beta), r, s-1\}\}$. Maintain $\delta_{i+1}(\beta)$, $\Lambda_{\beta,i+1}$, the current tree, and the current graph as in Case 2 of R1.

5.1.2. *Algorithms for* SHRINK. Suppose T_{h_u} is the equivalence-hanging tree with respect to $h_u = (L_0, L_1, \ldots, L_t)$. Let $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ be a maximal chain of $T^i_{h_u}$, where α_k is a leaf. Note that α_k is in a complete state and each $\alpha_j, 1 \leq j < k$, is in an almost-complete state without α_{j+1}. We define $level(\alpha_j) = q$ if $L_q \supseteq S_{\alpha_j}$. A node α_j in $\mathscr{C}$ is said to be a *jumped* 0*-node* if $\mathscr{C}$ contains a node α_s, $s \geq j$, such that $\gamma(\alpha_s) - (level(\alpha_s) - level(\alpha_j)) = 0$. Note that $\mathscr{C}$ may contain more than one jumped 0-node. We further say α_j is the *lowest jumped* 0*-node* if $\mathscr{C}$ contains no other jumped 0-node α_s with $s > j$.

LEMMA 12. *Suppose* $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ *is a chain. Let* $d_j = level(\alpha_j) - level(\alpha_1)$ *and* $l = \min\{\gamma(\alpha_1) - d_1, \gamma(\alpha_2) - d_2, \gamma(\alpha_3) - d_3, \ldots, \gamma(\alpha_k) - d_k\}$. *Then,* $cri(\alpha_1) = l$ *if one of the following two conditions is satisfied*: (1) $\gamma(u) \neq 0$ *and* $l \geq 1$, (2) $\gamma(u) = 0$ *and* $l \geq 0$.

Proof. We only consider the situation (1). The other one can be shown similarly. We show this lemma by induction on $length(\mathscr{C})$, the length of $\mathscr{C}$. We first consider $\mathscr{C} = [\alpha_1, \alpha_2]$ (i.e., $length(\mathscr{C}) = 1$). According to Algorithm SCD, if (α_2, α_1) is normal, then $cri(\alpha_1) = \min\{\gamma(\alpha_1), \gamma(\alpha_2) - 1\} = \min\{\gamma(\alpha_1) - d_1, \gamma(\alpha_2) - d_2\}$; otherwise, $cri(\alpha_1) = \min\{\gamma(\alpha_1), \gamma(\alpha_2)\} = \min\{\gamma(\alpha_1) - d_1, \gamma(\alpha_2) - d_2\}$, where $d_2 = level(\alpha_2) - level(\alpha_1) = 0$. Hence the basis case is true.

Assume the lemma is correct for $length(\mathscr{C}) < k - 1$. Assume that $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$. Here we consider (α_k, α_{k-1}) to be a normal edge. The case for (α_k, α_{k-1}) being abnormal can be proved similarly. Since α_{k-1} is in an almost-complete state without α_k, so $cri(\alpha_{k-1}) = \min\{\gamma(\alpha_{k-1}),$

$\gamma(\alpha_k) - 1\}$. Now the length of the resulting chain is less than $k - 1$. By the induction hypothesis,

$$\begin{aligned} cri(\alpha_1) &= \min\{\gamma(\alpha_1) - d_1, \gamma(\alpha_2) - d_2, \ldots, cri(\alpha_{k-1}) - d_{k-1}\} \\ &= \min\{\gamma(\alpha_1) - d_1, \gamma(\alpha_2) - d_2, \ldots, \\ &\qquad (\min\{\gamma(\alpha_{k-1}), \gamma(\alpha_k) - 1\}) - d_{k-1}\} \\ &= \min\{\gamma(\alpha_1) - d_1, \gamma(\alpha_2) - d_2, \ldots, \\ &\qquad \gamma(\alpha_{k-1}) - d_{k-1}, \gamma(\alpha_k) - 1 - d_{k-1}\} \\ &= \min\{\gamma(\alpha_1) - d_1, \gamma(\alpha_2) - d_2, \ldots, \\ &\qquad \gamma(\alpha_{k-1}) - d_{k-1}, \gamma(\alpha_k) - d_k\}. \end{aligned}$$

This completes the proof. ∎

With an argument similar to showing Lemma 12, we can generalize the above result as follows.

LEMMA 13. *Suppose* $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ *is a chain. Given an integer* s *such that* $1 \le s \le k$, *let* $d_j = level(\alpha_j) - level(\alpha_s)$ *for* $s \le j \le k$ *and let* $l = \min\{\gamma(\alpha_s) - d_s, \gamma(\alpha_{s+1}) - d_{s+1}, \ldots, \gamma(\alpha_k) - d_k\}$. *Then*, $cri(\alpha_s) = l$ *if one of the following two conditions is satisfied*: (1) $\gamma(u) \neq 0$ *and* $l \ge 1$, (2) $\gamma(u) = 0$ *and* $l \ge 0$.

LEMMA 14. *Suppose* $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ *is a chain and* α_t *is the lowest jumped* 0*-node of* $\mathscr{C}$. *Then*, α_t *is the largest-index node whose critical value is* 0.

Proof. Let $\Delta_j = \{\gamma(\alpha_j), \gamma(\alpha_{j+1}) - (level(\alpha_{j+1}) - level(\alpha_j)), \ldots, \gamma(\alpha_k) - (level(\alpha_k) - level(\alpha_j))\}$. Since α_t is the lowest jumped 0-node, $\min\{q|q \in \Delta_j\} > 0$ for $t < j \le k$, and $\min\{q|q \in \Delta_t\} = 0$. By Lemma 13, $cri(\alpha_t) = \min\{q|q \in \Delta_t\} = 0$ and $cri(\alpha_j) > 0$ for $t < j \le k$. Hence, the result holds. ∎

According to Lemma 14 and the computation of Algorithm SCD, we have the following lemma.

LEMMA 15. *Suppose* $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ *is a chain of* $T^i_{h_u}$ *with* $\gamma(u) = 0$ *and* α_t *being its lowest jumped* 0*-node. Then*, $cri(\alpha_1) = cri(\alpha_2) = \cdots = cri(\alpha_t) = 0$.

Below are two algorithms applied with SHRINK on $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$.

ALGORITHM S1. /* works on $\mathscr{C}$ when $\gamma_i(u) > 0$, where u is the root of the current hanging. */

Case 1. $\mathscr{C}$ contains a jumped 0-node. Find the lowest jumped 0-node α_t and compute its critical vertex set $\Lambda_{\alpha_t, i+1}$. If $t \neq k$, (α_{t+1}, α_t) is normal, and $cri(\alpha_{t+1}) = 1$, then compute the critical vertex set of α_{t+1}.

Let ω be an arbitrary node in $child_{T^0_{h_u}}(\alpha_t)$ satisfying $\gamma_i(\omega) = cri(\omega) = 1$ and (ω, α_t) is normal. Pick an arbitrary vertex x in $\Lambda_{\omega, i}$. Determine $tag(x)$ as in Case 1.1 of Algorithm SCD. If $tag(x) = 1$, let $\gamma_{i+1}(y) = 0$ for a vertex $y \in \Lambda_{\alpha_t, i+1}$ and execute Case 1.1.1 of Algorithm SCD to determine an equivalence-hanging tree T_{h_y}; otherwise, execute Case 1.1.2 of Algorithm SCD.

Case 2. $\mathscr{C}$ contains no jumped $0 =$ node. Let $d_j = level(\alpha_j) - level(\alpha_1)$ for $j = 1, \ldots, k$. Let $\gamma_{i+1}(\alpha_1) = cri(\alpha_1) = \min\{\gamma_i(\alpha_1) - d_1, \gamma_i(\alpha_2) - d_2, \gamma_i(\alpha_3) - d_3, \ldots, \gamma_i(\alpha_k) - d_k\}$ according to Lemma 12. Compute the critical vertex set $\Lambda_{\alpha_1, i+1}$ and let $\gamma_{i+1}(x) = cri(\alpha_1)$ for each $x \in \Lambda_{\alpha_1, i+1}$ if $\gamma_i(x) \neq cri(\alpha_1)$.

ALGORITHM S2. /* works on $\mathscr{C}$ when $\gamma(u) = 0$. */

Case 1. $\mathscr{C}$ contains a jumped 0-node. Find the lowest one α_t. Let $\gamma_{i+1}(\alpha_1) = \gamma_{i+1}(\alpha_2) = \cdots = \gamma_{i+1}(\alpha_t) = 0$ according to Lemma 15. Compute the critical vertex set $\Lambda_{\alpha_1, i+1}$ and let $\gamma_{i+1}(x) = 0$ for each $x \in \Lambda_{\alpha_1, i+1}$ if $\gamma_i(x) \neq 0$. Let $Z = \{\alpha_j | S_{\alpha_j}$ is an equivalence class, $1 \leq j \leq t\}$. For each $\alpha \in Z$, compute the critical vertex set $\Lambda_{\alpha, i+1}$ and let $\gamma_{i+1}(x) = 0$ for each $x \in \Lambda_{\alpha, i+1}$ if $\gamma_i(x) \neq 0$.

Case 2. $\mathscr{C}$ contains no jumped 0-node. The computation is the same as Case 2 of S1.

Remark. In Algorithm S2 and Case 2 of S1, we only need to compute the critical vertex set for the chain-leader α_1 and for those vertices whose γ-value are 0 (because they belong to a minimum connected γ-dominating set). In Case 1 of S1, we only need to compute the critical vertex sets of α_t and α_{t+1} (if $cri(\alpha_{t+1}) = 1$) for reconstructing a new hanging or determining a new minimum connected γ-dominating set. These are the reasons that Algorithms S1 and S2 do not compute the critical vertex sets for all the nodes of a maximal chain.

The following lemma provides a method to implement Case 2 of S1 and Case 2 of S2. The method can be used to implement Case 1 of S1 similarly.

LEMMA 16. *Suppose $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ is a chain of $T^i_{h_u}$ (not necessary maximal). If $cri(\alpha_j) > 0$, $2 \leq j \leq k$, the critical vertex set of α_1 can be found in $O(\log\log\log k)$ time using $O(k/\log\log\log k)$ processors on an arbitrary CRCW PRAM when $\mathscr{C}$ is reduced.*

Proof. There are two cases.

Case 1. (α_2, α_1) is normal. We first compute

$$\gamma_{i+1}(\alpha_1) = cri(\alpha_1)$$
$$= \min\{\gamma_i(\alpha_1), \gamma_i(\alpha_2) - (level(\alpha_2) - level(\alpha_1)), \ldots, \gamma_k(\alpha_k) - (level(\alpha_k) - level(\alpha_1))\}$$

in $O(\log\log\log k)$ time using $O(k/\log\log\log k)$ processors on a common CRCW PRAM [5]. If $\gamma_{i+1}(\alpha_1) < \delta_i(\alpha_1)$, then we select a vertex x from $\Lambda_{\alpha_1, i}$ and let $\Lambda_{\alpha_1, i+1} = \{x\}$; otherwise, $\Lambda_{\alpha_1, i+1} = \Lambda_{\alpha_1, i}$

Case 2. (α_2, α_1) is abnormal. We first compute the integer f so that $level(\alpha_1) = level(\alpha_2) = \cdots = level(\alpha_f)$ and $level(\alpha_{f+1}) = level(\alpha_f) + 1$. Then, we compute $cri(\alpha_f) = \min\{\gamma_i(\alpha_f), \gamma_i(\alpha_{f+1}) - (level(\alpha_{f+1}) - level(\alpha_f)), \ldots, \gamma_i(\alpha_k) - (level(\alpha_k) - level(\alpha_f))\}$. The above computation needs $O(\log\log\log k)$ time using $O(k/\log\log\log k)$ processors on a common CRCW PRAM [5]. Here we assume $f \neq k$. The case of $f = k$ can be implemented similarly. According to Lemma 13, we have $cri(\alpha_j) = \min\{\gamma_i(\alpha_j), \gamma_i(\alpha_{j+1}), \ldots, \gamma_i(\alpha_{f-1}), cri(\alpha_f)\}$, $1 \le j \le f$. Based on the suffix minimum finding technique[6] [5], $cri(\alpha_j)$ for all $1 \le j \le f$ can be computed in $O(\log\log\log k)$ time using $O(k/\log\log\log k)$ processors on an arbitrary CRCW PRAM.

Next, given the chain $[\alpha_f, \alpha_{f+1}, \ldots, \alpha_k]$, we compute the critical vertex set $\Lambda_{\alpha_{f,i+1}}$ using the method described in *Case* 1. Define a binary tree: for each node α_j, $1 \le j \le f-1$, let α_{j+1} represent its right child and $\Lambda_{\alpha_{j,i}}$ represent its left child. For each α_j, $1 \le j \le f-1$, if $cri(\alpha_{j+1}) \le \delta_i(\alpha_j)$, we mark α_{j+1}. If $cri(\alpha_{j+1}) \ge \delta_i(\alpha_j)$, we mark $\Lambda_{\alpha_{j,i}}$. The above operations need $O(1)$ time using $O(k)$ processors. We then find the smallest integer g between 1 and f so that α_g has no marked right child. If $f = g$, we also mark $\Lambda_{\alpha_{f,i+1}}$. Let $M(a, b) = \{\Lambda_{\alpha_j} | \Lambda_{\alpha_j}$ is marked, $a \le j \le b\}$ and $\Lambda(a, b) = \bigcup_{X \in M(a,b)} X$. If there exists no node α_j, $1 \le j \le g$, such that $\gamma_i(\alpha_j) < \min\{cri(\alpha_{j+1}), \delta_i(\alpha_j)\}$, then $\Lambda(1, g)$ is the critical vertex set of α_1; otherwise, let g' be the smallest index so that $\alpha_{g'}$ satisfies such a condition, and then $\{x\} \cup \Lambda(1, g'-1)$ is the critical vertex set of α_1, where x is an arbitrary vertex in $\Lambda(g', g)$. Since the union operation can be done easily by maintaining each set with a linked list, the desired parallel complexities can be achieved. ∎

[6] The *suffix minimum* of array $B = (b_1, b_2, \ldots, b_n)$, where b_i is an integer, is the elements of the array $(c_1, c_2, \ldots, c_n)$ such that $c_i = \min\{b_i, b_{i+1}, \ldots, b_n\}$, for $1 \le i \le n$.

We next show how to implement Case 1 of S2 to find the desired critical vertex sets. Using the method described in the proof of Lemma 16, we first find the critical vertex sets $\Lambda_{\alpha_1, i+1}$ and $\Lambda_{\alpha_t, i+1}$, respectively. For each $\alpha_j \in Z$, let $\alpha_{j'}$ denote the node in $\mathscr{C}$ satisfying $level(\alpha_j) = level(\alpha_{j+1}) = \cdots = level(\alpha_{j'}) \neq level(\alpha_{j'+1})$. If $\delta_i(\alpha_{j'}) > 0$, select an arbitrary vertex $x \in \Lambda_{\alpha_{j',i}}$ to be the critical vertex set of $\alpha_{j'}$; otherwise, $\Lambda_{\alpha_{j',i+1}} = \Lambda_{\alpha_{j',i}}$. We next compute the critical vertex set for each $\alpha_j \in Z$ as follows. Let $\mathscr{P} = \{\Lambda_{\alpha_{l,i}} | j \leq l < j' \text{ and } \delta_i(\alpha_l) = 0\}$. Let the union of the critical vertex set of $\alpha_{j'}$ and the set $\bigcup_{X \in \mathscr{P}} X$ be the critical vertex set of α_j. Note that the above implementation can be done in $O(\log\log\log k)$ time using $O(k/\log\log\log k)$ processors on a common CRCW PRAM.

In the rest of this section, we show how to find the lowest jumped 0-node in a maximal chain. We first partition the nodes of $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ according to their levels. This can be done by sorting in $O(\log k)$ time using $O(k)$ processors on an EREW PRAM [9]. Let $C_1, C_2, \ldots, C_l$ denote the partitioned sets. We define function $f: \{\alpha_1, \alpha_2, \ldots, \alpha_k\} \mapsto \{0, 1\}$ as follows. For each $C_s = [\alpha_{r_{s,1}}, \alpha_{r_{s,2}}, \ldots, \alpha_{r_{s,|C_s|}}]$, let $f(\alpha_{r_{s,1}}) = 1$ and let $f(\alpha_{r_{s,j}}) = 0$, $2 \leq j \leq |C_s|$. Let A be an array keeping $A[r_{s,j}] = f(\alpha_{r_{s,j}})$. We perform the parallel prefix sum computation [27] on A with $O(\log k)$ time using $O(k/\log k)$ processors on an EREW PRAM, and let $p_j = A[j]$ for all $1 \leq j \leq k$, after this computation. We define a new level function l with $l(\alpha_j) = p_j$, $1 \leq j \leq k$. For ease of referencing, we call the above work *the preprocessing of* $\mathscr{C}$. A node $\alpha_p \in \mathscr{C}$ is a jumped 0-node if there is a node α_q for $q \geq p$ with $\gamma(\alpha_q) - (l(\alpha_q) - l(\alpha_p)) = 0$. For each $\alpha_j \in \mathscr{C}$, let $weight(\alpha_j) = l(\alpha_j) - \gamma(\alpha_j)$. We then compute $x = \max\{weight(\alpha_1), weight(\alpha_2), \ldots, weight(\alpha_k)\}$ in $O(\log\log\log k)$ time using $O(k/\log\log\log k)$ processors on a common CRCW PRAM [5]. If $x > 0$, then $\alpha_{r_{x,|C_x|}}$ is the lowest jumped 0-node. If $x \leq 0$, then $\mathscr{C}$ contains no jumped 0-node.

LEMMA 17. *Given a maximal chain* $\mathscr{C} = [\alpha_1, \alpha_2, \ldots, \alpha_k]$ *after preprocessing, the lowest jumped* 0*-node can be computed in* $O(\log\log\log k)$ *time using* $O(k/\log\log\log k)$ *processors on a common CRCW PRAM.*

5.2. *The Complete Parallel Algorithm and Its Implementation*

Before preceding to the description of our complete parallel algorithm, PCD, we first describe a method to preprocess an equivalence-hanging tree to make our implementation more efficient. Given T_{h_u}, we make a copy of this tree. We then contract it in $t \in O(\log n)$ phases using the strategy described in Section 3. With the help of tree contraction, we find subsets of the nodes of T_{h_u} that are leaves or maximal chains in each contraction phase of Algorithm $R \& S$. Then, for the nodes deleted by RAKE, we

further partition them into several common-parent leaf sets according to their parents. For each subset $\mathscr{C}$ corresponding to a maximal chain, we perform the list raking [27] to number the nodes of $\mathscr{C}$ starting from the chain-leader and preprocess it by the argument to show Lemma 17. Hence, all the maximal chains and common-parent leaf sets (with respect to the ith contraction phase for all $1 \le i \le t$) can be completely identified in $O(\log n \cdot \log\log\log n)$ time using $O(n \log\log\log n)$ processors on an arbitrary CRCW PRAM.

For each $v \in V(T_{h_u})$, the initial $\gamma(v)$ and the vertex set Λ_v can be computed in $O(\log n)$ time using $O(n)$ processors on an EREW PRAM based on Cole's parallel sorting [9] and minimum finding technique [27]. Throughout this implementation, we assume Λ_v is manipulated with a linked list. We now present our complete parallel algorithm to solve the minimum connected γ-dominating set problem.

ALGORITHM PCD

INPUT: A connected distance-hereditary graph $G = (V, E)$.
OUTPUT: A minimum connected γ-dominating set $D(G)$.
(1) **if** there is a vertex $u \in V$ with $\gamma(u) = 0$
(2) determine the hanging h_u;
(3) $flag := 0$;
(4) **else** determine a hanging rooted at an arbitrary vertex, u;
(5) $flag := 1$;
(6) **endif**
(7) construct an equivalence-hanging tree T_{h_u} and preprocess it;
(8) **While** $T^i_{h_u}$ is not a single vertex **do**
(9) let $W_1, W_2, \ldots, W_k$ be all the maximal common-parent leaf sets of $T^i_{h_u}$;
(10) **if** there is a leaf $\alpha \in W_j$ satisfying Case 1 of R1
(11) perform R1(W_j) to generate $D(G)$ and terminate, or reconstruct an equivalence-hanging tree (along with preprocessing);
(12) **if** a new equivalence-hanging tree is constructed
(13) $flag := 0$ and goto line (8);
(14) **endif**
(15) **endif**
(16) **else**
(17) **for** each W_j **do in parallel**
(18) **if** $flag = 1$ then perform R1(W_j);
(19) **else** perform R2(W_j);
(20) **endif**
(21) remove W_j from $T^i_{h_u}$;
(22) **endfor**

(23) let $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_k$ be all the maximal chains of $T^i_{h_u}$;
(24) **if** there is a $\mathscr{C}_j$ satisfying the condition of Case 1 of S1
(25) perform S1($\mathscr{C}_j$) to generate $D(G)$ and terminate, or reconstruct a new equivalence-hanging tree (along with preprocessing);
(26) **if** a new equivalence-hanging tree is constructed
(27) *flag* := 0; goto line (8);
(28) **endif**
(29) **endif**
(30) **else**
(31) **for** each $\mathscr{C}_j$ **do in parallel**
(32) **if** *flag* = 1 then perform S1($\mathscr{C}_j$);
(33) **else** perform S2($\mathscr{C}_j$);
(34) **endif**
(35) reduce $\mathscr{C}_j$ from $T^i_{h_u}$;
(36) **endfor**
(37) **endwhile**
/* T_{h_u} *is reduced to its root.* */
(38) **if** *flag* = 0, then $D(G) = \{z \in V | \gamma(z) = 0\} = \bigcup_{\alpha \in \mathscr{S}} \Lambda_\alpha$, where $\mathscr{S} = \{\alpha | cri(\alpha) = 0$ and S_α represents an equivalence class$\}$;
(39) **else** $D(G) = \{u\}$, where u in the root of the given hanging;
(40) **end of the algorithm**.

The correctness of PCD can be shown by induction on the number of contraction phases.

We now show how to implement the algorithm in $O(\log n \cdot \log\log\log n)$ time using $O((n+m)/\log\log\log n)$ processors on an arbitrary CRCW PRAM. By Lemma 10 and the parallel strategy to compute a hanging [13] and preprocess T_{h_u}, lines 1–7 can be done in $O(\log n \cdot \log\log\log n)$ time using $O((n+m)/\log\log\log n)$ processors on an arbitrary CRCW PRAM. By Theorem 1, we can preprocess T_{h_u} to find all the chains in each tree contraction stage in $O(\log n \cdot \log\log\log n)$ time using $O((n+m)/\log\log\log n)$ processors on an arbitrary CRCW PRAM.

According to Lemmas 1 and 6, the iteration at lines 9–37 is executed in $O(\log n)$ times. In each iteration, lines 9–22 (corresponding to RAKE) can be implemented as follows. We first consider the situation that $\gamma(u) \neq 0$, where u is the root of the current hanging. We can decide which case of R1 to be applied in constant time. Suppose that the condition of Case 1 of R1 holds, the tag value of vertex x can be determined by computing the hanging rooted at x. It takes $O(\log n)$ time using $O(n+m)$ processors on an arbitrary CRCW PRAM [13]. If $tag(x) = 0$, then output $D(G)$. If $tag(x) = 1$, then we construct a new equivalence-hanging tree. The above work can be done by executing line 11 in $O(\log n)$ time using $O(n+m)$ processors on an arbitrary CRCW PRAM by Lemma 10. Now assume the

condition of Case 2 of R1 holds. Line 18 is performed on each maximal common-parent leaf set. Note that computing γ-values and union of the given sets can be done in $O(\log\log\log n)$ time using $O(n/\log\log\log n)$ processors on a common CRCW PRAM [5].

We now consider the case when $\gamma(u) = 0$, line 19 (corresponding to Algorithm R2) is executed in $O(\log\log\log n)$ time using $O(n/\log\log\log n)$ processors on a common CRCW PRAM. Note that generating $D(G)$, reconstructing an equivalence-hanging tree, and determining the tag value is executed at most one time under $O(\log n)$ contraction phases. Therefore, the execution of lines 9–22 totally takes $O(\log n \cdot \log\log\log n)$ time using $O((m + m)/\log\log\log n)$ processors on an arbitrary CRCW PRAM after $O(\log n)$ phases.

We now consider the implementation of lines 23–36 in each iteration. We first consider the case when $\gamma(u) \neq 0$. For all the maximal chains $\mathscr{C}_j$ (after preprocessing) in the current tree, their lowest jumped 0-nodes can be computed in $O(\log\log\log n)$ time using $O(n/\log\log\log n)$ processors on an arbitrary CRCW PRAM according to Lemma 17. If no maximal chain contains jumped 0-node, Case 2 of S1 (corresponding to line 32) can be implemented in $O(\log\log\log n)$ time using $O(n/\log\log\log n)$ processors on an arbitrary CRCW PRAM by Lemmas 16 and 17 and minimum finding technique [5]; otherwise, we find one maximal chain $\mathscr{C}_j$ and its lowest jumped 0-node to either output $D(G)$ or reconstruct a new equivalence-hanging tree (line 25). It takes $O(\log n)$ time using $O(n + m)$ processors on an arbitrary CRCW PRAM.

Now we consider the case when $\gamma(u) = 0$. It is not difficult to see that the desired complexity can be achieved. Hence, lines 23–36 (corresponding to SHRINK) can be implemented in $O(\log n \cdot \log\log\log n)$ time using $O((n + m)/\log\log\log n)$ processors on an arbitrary CRCW PRAM after $O(\log n)$ phases. Note that checking whether the current tree is a single vertex and setting *flag* can be done easily. Besides, the implementation of lines 38 and 39 can be done in constant time. With the aid of Brent's scheduling principle, we conclude this section with the following result.

THEOREM 3. *The minimum connected γ-dominating set problem on distance-hereditary graphs can be solved by Algorithm* PCD *in* $O(\log n \cdot \log\log\log n)$ *time using* $O((n + m)/\log\log\log n)$ *processors on an arbitrary CRCW PRAM.*

6. THE MINIMUM γ-DOMINATING CLIQUE PROBLEM

A linear time sequential algorithm is first described in [16]. Here we present another sequential algorithm which can help us to design a

parallel algorithm. We show that the minimum γ-dominating clique problem on distance-hereditary graphs can also be solved using $R\,\&\,S$. Let G' be an induced subgraph of G with a new γ value γ' assigned to each of its vertices. Let $\mathscr{K}(G)$ and $\mathscr{K}(G')$ denote a minimum γ-dominating clique of G and a minimum γ'-dominating clique of G', respectively. The sequential algorithm for finding a minimum γ-dominating clique on a distance-hereditary graph G is also based on a reduction scheme similar to the one described in Fact 4. That is, we first properly choose G' and set γ', and then reduce the problem of computing $\mathscr{K}(G)$ to the problem of computing $\mathscr{K}(G')$.

Fact 6 [16, 17]. Let $\mathscr{Q}$ be a vertex subset of the graph $G = (V, E)$.

(a) Assume that $\mathscr{Q} \subset V$ is a proper homogeneous set of G. Let x be a vertex of $\mathscr{Q}$ such that $\gamma(x) = \min\{\gamma(y) | y \in Q\}$. Also let $G' = \langle (V \setminus \mathscr{Q}) \cup \{x\} \rangle$ and $\gamma'(v) = \gamma(v)$ for all $v \in G'$. If $\gamma(x) \geq 2$, or $\gamma(x) = 1$ and $tag(x) = 1$, then $\mathscr{K}(G) = \mathscr{K}(G')$.

(b) Assume that $\mathscr{Q}$ contains only one vertex x such that $N(x)$ forms a homogeneous set in G. Let y be a vertex of $N(x)$ with $\gamma(y) = \min\{\gamma(z) | z \in N(x)\}$. Also let $G' = \langle V \setminus \{x\} \rangle$ and $\gamma'(v) = \gamma(v)$ for all $v (\neq y) \in G'$. If $\gamma(x) \geq 2$, or $\gamma(x) = 1$ and $tag(x) = 1$, then $\mathscr{K}(G) = \mathscr{K}(G')$ with $\gamma'(y) = \min\{\gamma(y), \gamma(x) - 1\}$.

The construction of an equivalence-hanging tree T_{h_u} with associated data structure, and the method to reduce it are the same as Algorithms SCD in Section 4.3. Let α be an arbitrary leaf of $T^i_{h_u}$ which is not the root. Also let $par(\alpha) = \beta$. We consider the following two cases to process α.

Case 1. $\gamma_i(u) > 0$, where u is the root of the current hanging.

Case 1.1. $\gamma_i(\alpha) = 0$. In this case, there exists a node ω in $Nchild_{T^0_{h_u}}(\alpha)$ with $\gamma_i(\omega) = 1$. /* *ω is not in $T^i_{h_u}$.* */ Pick an arbitrary vertex x in $\Lambda_{\omega, i}$. Let $tag(x) = 1$ if there is a vertex $v \in V(G^i)$ with $dist_G(x, v) > \gamma_i(v)$. Otherwise, let $tag(x) = 0$.

Case 1.1.1. $tag(x) = 1$. Pick a vertex $y \in \Lambda_{\alpha, i}$. /* Note that $\gamma_i(y) = 0$. */ Let G' denote the subgraph of G induced by $(\bigcup_{\nu \in V(T^0_{h_u}) \setminus V(T^0_{T_{h_u}(\alpha)})} S_\nu) \cup S_\alpha$. Let $\gamma_{i+1}(v) = \gamma_0(v)$ for all $v \in V(G') \setminus S_\alpha$. Determine the hanging h_y of G'. Construct the equivalence-hanging tree T_{h_y}. Replace $T^i_{h_u}$ with T_{h_y}, and goto the next iteration.

Case 1.1.2. $tag(x) = 0$. /* *x γ-dominates all vertices of $V(G) \setminus S_\omega$.* */ If there is a vertex $w \in S_\omega$ satisfying $N[w] \supseteq \Lambda_{\omega, i}$, output $\mathscr{K}(G) = \{w\}$ and terminate the execution; otherwise, output $\mathscr{K}(G) = \{x, z\}$, where z is an arbitrary vertex in $N'(S_\omega)$, and terminate the execution.

Case 1.2. $\gamma_i(\alpha) \geq 1$.

Case 1.2.1. (α, β) is a normal edge. Let $\gamma_{i+1}(\beta) = \min\{\gamma_i(\alpha) - 1, \gamma_i(\beta)\}$. Remove α and S_α from $T^i_{h_u}$ and $V(G^i)$, respectively. If α is the only child of β in $T^i_{h_u}$ and $\gamma_{i+1}(\beta) < \delta_i(\beta)$, then pick an arbitrary vertex g in $\Lambda_{\beta,i}$, let $\gamma_{i+1}(g) = \gamma_{i+1}(\beta)$, and $\Lambda_{\beta,i+1} = \{g\}$.

Case 1.2.2. (α, β) is an abnormal edge. Let $\gamma_{i+1}(\beta) = \min\{\gamma_i(\alpha), \gamma_i(\beta)\}$. If $\gamma_i(\alpha) = \delta_i(\beta)$, then $\Lambda_{\beta,i+1} = \Lambda_{\alpha,i} \cup \Lambda_{\beta,i}$. If $\gamma_i(\alpha) < \delta_i(\beta)$, then let $\delta_{i+1}(\beta) = \delta_i(\alpha)$ and $\Lambda_{\beta,i+1} = \Lambda_{\alpha,i}$. Remove α from $T^i_{h_u}$. If α is the only child of β in $T^i_{h_u}$ and $\gamma_{i+1}(\beta) < \delta_{i+1}(\beta)$, then pick an arbitrary vertex g in $\Lambda_{\beta,i+1}$, let $\gamma_{i+1}(g) = \gamma_{i+1}(\beta)$, and $\Lambda_{\beta,i+1} = \{g\}$.

Case 2. $\gamma_i(u) = 0$.

If $\gamma_i(\alpha) = 0$ and β is not the root, then our algorithm terminates. /* *The current graph contains no γ-dominating clique because there are two vertices x and y satisfying $\gamma(x) = \gamma(y) = 0$ and dist$(x, y) > 1$.* */ Otherwise, use the same method as Case 1.2 to maintain $\gamma_{i+1}(\beta)$, $\delta_{i+1}(\beta)$, $\Lambda_{\beta,i+1}$, the current tree, and the current graph.

Our sequential algorithm, called SDK, works by applying the above two cases to T_{h_u} until T_{h_u} is reduced to its root or until a minimum γ-dominating clique is found by executing Case 1.1.2. Suppose T_{h_u} is reduced to its root. If there is no vertex whose γ-value is adjusted to 0 during the execution, then the root of the hanging is a minimum γ-dominating clique; otherwise, G contains a minimum γ-dominating clique $H = \{w \in N[u] \| \gamma(w) = 0\}$ if $\langle H \rangle$ forms a complete subgraph. Clearly, the structure of Algorithm SDK is similar to that of Algorithm SCD. By slightly modifying Algorithms R1, R2, S1, and S2 developed in Section 5, we can also parallelize SDK with the desired complexities.

THEOREM 4. *The minimum γ-dominating clique problem on distance hereditary graphs can be solved in $O(\log n \cdot \log\log\log n)$ time using $O((n + m)/\log\log\log n)$ processors on an arbitrary CRCW PRAM.*

7. DISCUSSION AND CONCLUSION

We have proposed a new implementation of the tree contraction scheme *R & S*. Based on this scheme, we have solved the minimum connected γ-dominating set and minimum γ-dominating clique problems in parallel on distance-hereditary graphs. Furthermore, both γ-dominating set problems can be solved in $O(\log n \cdot T(n))$ time using $O(P(n) + (n + m)/T(n))$

processors on an arbitrary CRCW PRAM, where $T(n)$ and $P(n)$ are the time and processor complexities required to solve the maximum finding and all the nearest smaller values computing problems. Dragan [16] showed that the problems of finding a central vertex, finding a central clique, and computing the radius on distance-hereditary graphs G can be solved using the algorithm to compute a minimum γ-dominating clique $\mathscr{K}(G)$. The key concept is first to set special γ-values on $V(G)$ and then run the algorithm to compute $\mathscr{K}(G)$. Since the transformation can be easily parallelized, the above mentioned problems can be solved with the same time-processor complexity as solving the γ-dominating clique problem. Our results show these problems on distance-hereditary graphs belonging to *NC* class, i.e., the class of problems which can be solved by parallel random access machines in polylogarithmic parallel time with polynomial many processors [28]. We hope that our general parallel technique can be applied to other special classes of graphs which are tree-representable.

ACKNOWLEDGMENT

We thank the anonymous referees for providing helpful information and pointing out several related references of the tree contraction scheme used in the paper. We also thank one anonymous referee for comments that improved the paper.

REFERENCES

1. K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka, A simple parallel tree contraction algorithm, *J. Algorithms* **10** (1989), 287–302.
2. H. J. Bandelt and H. M. Mulder, Distance-hereditary graphs, *J. Combin. Theory Ser. B* **41** (1989), 182–208.
3. C. Berge, "Graphs and Hypergraphs," North-Holland, Amsterdam, 1973.
4. O. Berkman, B. Schieber, and U. Vishkin, Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values, *J. Algorithms* **14** (1993), 344–370.
5. O. Berkman, Y. Matias, and P. Ragde, Triply-logarithmic parallel upper and lower bounds for minimum and range minimum over small domains, *J. Algorithms* **28** (1998), 197–215.
6. A. Brandstadt, "Special Graph Classes—A Survey," Technical Report SM-DU-199, University of Duisburg, 1993.
7. A. Branstadt and F. F. Dragan, A linear time algorithm for connected γ-domination and Steiner tree on distance-hereditary graphs, *Networks* **31** (1998), 177–182.
8. G. J. Chang, Labeling algorithms for domination problems in sun-free chordal graphs, *Discrete Appl. Math.* **22** (1988), 21–34.
9. R. Cole, Parallel merge sort, *SIAM J. Comput.* **17** (1988), 770–785.

10. R. Cole and U. Vishkin, Approximate parallel scheduling, II: application to optimal parallel graph algorithms in logarithmic time, *Inform. and Comput.* **92** (1991), 1–47.
11. E. Dahlhaus, Optimal (parallel) algorithms for the all-to-all vertices distance problem for certain graph classes, *in* "Proceedings of 18th International Workshop on Graph-Theoretic Concepts in Computer Science," Lecture Notes in Computer Science, Vol. 657, pp. 60–69, Springer-Verlag, Berlin, 1993.
12. E. Dahlhaus and P. Damaschke, The parallel solution of domination problems on chordal and strongly chordal graphs, *Discrete Appl. Math.* **52** (1994), 261–273.
13. E. Dahlhaus, Efficient parallel recognition algorithms of cographs and distance-hereditary graphs, *Discrete Appl. Math.* **57** (1995), 29–44.
14. A. Dátri and M. Moscarini, Distance-hereditary graphs, steiner trees, and connected domination, *SIAM J. Comput.* **17** (1988), 521–538.
15. F. F. Dragan, HT-graphs: centers, connected γ-domination and Steiner trees, *Comput. Sci. J. Moldova* **1** (1993), 64–83.
16. F. F. Dragan, Dominating cliques in distance-hereditary graphs, *in* "Proceedings of the 4th Scandinavian Workshop on Algorithm Theory," Lecture Notes in Computer Science, Vol. 824, pp. 370–381, Springer-Verlag, Berlin, 1994.
17. F. F. Dragan and A. Brandstadt, γ-dominating cliques in Helly graphs and chordal graphs, *in* "Proceedings of 11th Theoretical Aspects of Computer Science," Lecture Notes in Computer Science, Vol. 775, pp. 735–746, Springer-Verlag, Berlin, 1994.
18. H. Gazit, G. L. Miller, and S. H. Tseng, Optimal tree contraction in the EREW model, *in* "Princeton Workshop Book," Plenum, New York, 1989.
19. A. Gibbons and W. Rytter, Optimal parallel algorithms for dynamic expression evaluation and context-free recognition, *Inform. and Comput.* **81** (1989), 32–45.
20. M. C. Golumbic, "Algorithmic Graph Theory and Perfect Graphs," Academic Press, New York, 1980.
21. P. L. Hammer and F. Maffray, Complete Separable Graphs, *Discrete Appl. Math.* **27** (1990), 85–99.
22. X. He, Efficient parallel algorithms for solving some tree problems, *in* "Proc. 24th Allerton Conference on Communication, Control, and Computing," pp. 777–786, 1986.
23. X. He and Y. Yesha, Binary tree algebraic computation and parallel algorithms for simple graphs, *J. Algorithms* **9** (1988), 92–113.
24. S. C. Hedetniemi and R. Laskar (Eds.), Topics on domination, *in* "Annals of Discrete Mathematics," p. 48, North-Holland, Amsterdam, 1991.
25. E. Howorka, A characterization of distance-hereditary graphs, *Quart. J. Math.* (*Oxford*), **28** (1977), 417–420.
26. 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), 43–52. [A preliminary version of this paper is in "Proceedings of the International Conference on Parallel Processing," pp. 20–23, 1997.]
27. J. JáJá, "An Introduction to Parallel Algorithms," Addison-Wesley, Reading, MA, 1992.
28. R. M. Karp and V. Ramachandran, Parallel algorithms for shared memory machines, *in* "Handbook of Theoretical Computer Science," pp. 869–941, North-Holland, Amsterdam, 1990.
29. P. Klein, Efficient parallel algorithms for chordal graphs, *SIAM J. Comput.* **25** (1996), 797–827.
30. G. L. Miller and J. Reif, Parallel tree contraction and its applications, *in* "Proceedings of 17th Annual Symposium on the Theory of Computing," pp. 478–489, 1985.
31. G. L. Miller and J. Reif, Parallel tree contraction, Part 2: Further applications, *SIAM J. Computing* **20** (1991), 1128–1147.

32. J. Naor, M. Naor, and A. A. Schäffer, Fast parallel algorithms for chordal graphs, *SIAM J. Comput.* **18** (1989), 327–349.
33. V. Ramachandran, Fast parallel algorithms for reducible flow graphs, *in* "Concurrent Computations: Algorithms, Architecture and Technology" (S. K. Tewksbury, B. W. Dickson, and S. C. Schwartz, Eds.), Proc. of the 1987 Princeton Workshop on Algorithm, Architecture and Technology Issues for Models of Concurrent Computations, pp. 117–138, Plenum Press, New York, 1988.
34. V. Ramachandran, Parallel algorithms for reducible flow graphs, *J. Algorithms* **23** (1997), 1–31.