Available online at www.sciencedirect.com
SCIENCE DIRECT
Discrete Applied Mathematics 154 (2006) 508–524
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

The Hamiltonian problem on distance-hereditary graphs☆

Sun-Yuan Hsieh[a,1], Chin-Wen Ho[b,2], Tsan-Sheng Hsu[c], Ming-Tat Ko[c]

[a]*Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan*
[b]*Department of Computer Science and Information Engineering, National Central University, Chung-Li, Taiwan*
[c]*Institute of Information Science, Academia Sinica, Taipei, Taiwan*

Received 19 November 2002; received in revised form 25 May 2005; accepted 20 July 2005
Available online 5 October 2005

Abstract

In this paper, we first present an $O(n+m)$-time sequential algorithm to solve the Hamiltonian problem on a distance-hereditary graph G, where n and m are the number of vertices and edges of G, respectively. This algorithm is faster than the previous best known algorithm for the problem which takes $O(n^2)$ time. We also give an efficient parallel implementation of our sequential algorithm. Moreover, if G is represented by its decomposition tree form, the problem can be solved optimally in $O(\log n)$ time using $O((n+m)/\log n)$ processors on an EREW PRAM.

Keywords: Distance-hereditary graphs; The Hamiltonian problem; Parallel algorithms; PRAM

1. Introduction

A graph is *distance-hereditary* [2,11] if the distance stays the same between any 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 [7,10,11] that are graphs G in which the maximum clique size equals the chromatic number for every induced subgraph of G [3,9]. Two well-known classes of graphs, trees and cographs, both belong to this subclass of distance-hereditary graphs. The properties of distance-hereditary graphs have been the focus of great deal of research [2,4,5,7,8,10–16,20,22,25–28], which has resulted in sequential or parallel algorithms for solving several interesting graph-theoretical problems of this special class of graphs.

A cycle in a graph G is called a *Hamiltonian cycle* if it contains every vertex of G exactly once. A graph is said to be *Hamiltonian* if it contains a Hamiltonian cycle. The *Hamiltonian problem* is to determine whether there exists a Hamiltonian cycle in a given graph and then find one if such a cycle does exist. Previous related works on distance-hereditary graphs are summarized below. By investigating the neighborhood of the last pendant vertex in a one-vertex extension-sequence, Müller and Nicolai [20] developed an $O(n(n+m))$-time sequential algorithm to

☆A preliminary version of this paper appeared in Proceedings of the Eigth Annual International Conference on Computing and Combinatorics (COCOON 2002), Lecture Notes in Computer Science 2387, 2002, pp. 77–86.

E-mail addresses: hsiehsy@mail.ncku.edu.tw (S.-Y. Hsieh), hocw@csie.ncu.edu.tw (C.-W. Ho), tshsu@iis.sinica.edu.tw (T.-S. Hsu), mtko@iis.sinica.edu.tw (M.-T. Ko).

[1]Part of this work was carried out while this author was visiting the Institute of Information Science, Academia Sinica, Taipei, Taiwan.

[2]The work of this author was supported in part by National Science Council under grant NSC91-2213-E-008-034.

0166-218X/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.07.012

solve the Hamiltonian problem on bipartite distance-hereditary graphs, where n, (respectively, m) is the number of vertices (respectively, edges) of the given graph. In [22], Nicolai presented an $O(n^3)$-time sequential algorithm to solve the Hamiltonian problem on distance-hereditary graphs. Hung et al. [16] proposed an $O(n^2)$-time sequential algorithm to solve the problem. Quite recently, similar parallel algorithms for other perfect graphs have been proposed. Nakano et al. [21] constructed a path-tree for cographs (a subclass of distance-hereditary graphs [6]) and utilized this data structure to solve the path cover and Hamiltonian path problems on cographs. They used $O(T_c(n, m) + \log n)$ time using $O(P_c(n, m) + (n + m)/\log n)$ processors on M_c, where $T_c(n, m)$ and $P_c(n, m)$, respectively denote the parallel time and processor complexities required to construct a cotree representation of a cograph on a PRAM model M_c. Several parallel algorithms to construct a cotree that run in $O(\log^2 n)$ time were known [6,24]. The algorithm in [6] used $O(n + m)$ CREW processors. Recently, Nikolopoulos and Palios [24] devised an EREW algorithm that uses $O((n + m)/\log n)$ processors. On the other hand, Nikolopoulos [23] solved the Hamiltonian problem on quasi-threshold graphs (a subclass of cographs [23], and therefore a subclass of distance-hereditary graphs) in $O(\log n)$ time using $O(n + m)$ processors on a CREW PRAM. To sum up, [21] and [23] discovered tree-like structures to achieve parallelization of their solutions. We discover additional insights in another tree-like structure. The structure is simpler and hence we can obtain our solution by applying the well known tree contraction technique on it (see [1]).

In this paper, we adopt a different approach to solve the Hamiltonian problem on distance-hereditary graphs. We first present an $O(n + m)$-time algorithm for solving the problem. Let $T_d(n, m)$ and $P_d(n, m)$, respectively denote the parallel time and processor complexities required to construct a decomposition tree representation of a distance-hereditary graph on a PRAM model M_d. We show that the Hamiltonian problem can be solved in $O(T_d(n, m) + \log n)$ time using $O(P_d(n, m) + (n + m)/\log n)$ processors on M_d. The best known method for constructing a decomposition tree needs $O(\log^2 n)$ time using $O(n + m)$ processors on a CREW PRAM [12,15]. If G is given in its decomposition tree form, the problem can be solved in $O(\log n)$ time using $O((n + m)/\log n)$ processors on an EREW PRAM. This time-processor complexity matches the best sequential algorithm. The parallel 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 [18] (see also [17]).

The rest of this paper is organized as follows. In Section 2, we review some properties of distance-hereditary graphs and give some basic definitions. In Section 3, we present a sequential linear-time algorithm to solve the Hamiltonian problem on distance-hereditary graphs. In Section 4, we provide an efficient parallel implementation of our sequential algorithm. Finally, in Section 5, we present our concluding remarks.

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-theoretical terminologies and notations not mentioned here, readers should refer to [9].

Bandelt and Mulder [2] showed that the house, holes, domino, and gem are neither induced subgraphs nor isometric subgraphs of a distance-hereditary graph. They also showed that every finite distance-hereditary graph G has a one-vertex-extension ordering which can generate G from K_1 by iterating the following two operations: adding pendant vertices and splitting vertices [2]. Chang et al. [5] generalized the concept of the one-vertex-extension ordering to define the one-vertex-extension tree. According to this tree, they further showed that each distance-hereditary graph G is associated with a subset of vertices, called a *twin set*, and that G can be constructed from two distance-hereditary graphs by three operations working on corresponding twin sets, which are formally described in Theorem 1. Interested readers may consult [5] for the proof of Theorem 1.

Theorem 1 (*Chang et al. [5]*). *A graph is distance-hereditary if and only if it has the following recursive construction*:
A graph consisting of a single vertex v is a primitive distance-hereditary graph with twin set $\{v\}$. Let G_1 and G_2 be distance-hereditary graphs with twin sets S_1 and S_2, respectively. Then,

(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 twin set $S_1 \cup S_2$*;

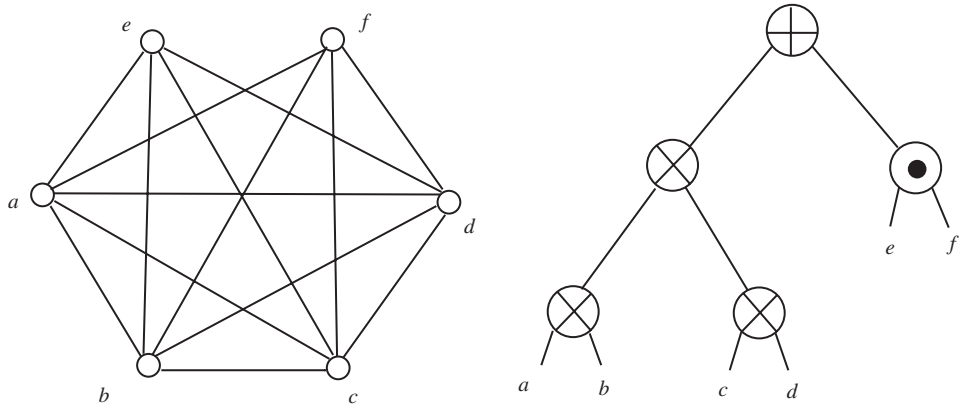

Fig. 1. A distance-hereditary graph and its decomposition tree.

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

Remarks. We note here that a distance-hereditary graph G can have more than one twin set. Different constructions of G using the three operations defined in Theorem 1 may have different twin sets. The correctness of our method is based on a recursive construction of G and its corresponding twin set. It does not depend on the uniqueness of twin set.

A distance-hereditary graph G is said to be formed from G_1 and G_2 by the *true twin*, *attachment*, and *false twin* operations if G is obtained from (1)–(3) of Theorem 1, respectively.

For a rooted tree T, we use $root(T)$ to denote the root of T. A distance-hereditary graph can be represented by a binary tree form, called a *decomposition tree* T_G, which is defined as follows.

Definition 1 (*Chang et al. [5]*). (1) The tree consisting of a single vertex v is a decomposition tree of a primitive distance-hereditary graph $G = (\{v\}, \emptyset)$.

(2) Let T_{G_1} and T_{G_2} be 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 T_G is a decomposition tree of G if $root(T_G)$ is represented by $\otimes$, and $root(T_{G_1})$ and $root(T_{G_2})$ are the two children of $root(T_G)$.
(b) If G is formed from G_1 and G_2 by the attachment operation, then T_G is a decomposition tree of G if $root(T_G)$ is represented by $\oplus$, and $root(T_{G_1})$ and $root(T_{G_2})$, respectively are the left and right children of $root(T_G)$.
(c) If G is formed from G_1 and G_2 by the false twin operation, then T_G is a decomposition tree of G if $root(T_G)$ is represented by $\odot$, and $root(T_{G_1})$ and $root(T_{G_2})$ are the two children of $root(T_G)$.

Fig. 1 shows a distance-hereditary graph and its decomposition tree. Note that the twin set of the graph is $\{a, b, c, d\}$.

For a node v in T_G, let $T_G(v)$ be the subtree of T_G rooted at v, and let G_v be a subgraph of G induced by the leaves of $T_G(v)$. Also let S_v be the twin set of G_v.

Lemma 1 (*Chang et al. [5]*). *A decomposition tree of a distance-hereditary graph can be constructed in* $O(n + m)$ *sequential time*.

Lemma 2 (*Hsieh [12], Hsieh et al. [15]*). *A decomposition tree of a distance-hereditary graph can be constructed in* $O(\log^2 n)$ *time using* $O(n + m)$ *processors on a CREW PRAM*.

Remarks. Dalhaus [6] developed a parallel recognition algorithm for distance-hereditary graphs that takes $O(\log^2 n)$ time using $O(n + m)$ processors on a CREW PRAM, without constructing decomposition trees. Hsieh [12] utilized

the cotree construction algorithm described in [6] to construct a decomposition tree for a distance-hereditary graph in $O(\log^2 n)$ time using $O(n+m)$ processors on a CREW PRAM.

A *closed integer interval* is an ordered pair of integers $[t_1, t_2]$, with $t_1 \leqslant t_2$. The interval $[t_1, t_2]$ represents the set $\{t \in \mathbf{Z} | t_1 \leqslant t \leqslant t_2\}$. A *path partition* of a graph $G = (V, E)$ is a set of pairwise vertex disjoint paths such that the union of the vertices of these paths equals V. Given a distance-hereditary graph G with the twin set S, a path partition of G is said to be *crucial* if the end-vertices of each path are in S. We denote a *crucial path partition* of G by $\mathscr{P}_S(G)$. Furthermore, a *crucial k-path partition* of G, $\mathscr{P}_S^k(G) = \{P_1, P_2, \ldots, P_k\}$, is a crucial path partition of G composed exactly of k paths $P_1, P_2, \ldots, P_k$. Throughout this paper all path partitions are considered crucial, and the subscripts S in the notations $\mathscr{P}_S(G)$ and $\mathscr{P}_S^k(G)$ are omitted if no ambiguity arises. Let $N(G) = (l_1, l_2, \ldots, l_t)$ denote the set of integers in increasing order, i.e., $l_1 < l_2 < \cdots < l_t$, such that G has a crucial l_i-path partition. As we show in Section 3.1, if G is a distance-hereditary graph and contains a crucial path partition, then the elements of $N(G)$ form a segment of consecutive integers, that is, $l_{i+1} = l_i + 1$ for all $1 \leqslant i \leqslant t-1$. Thus $N(G)$ can be represented by a closed integer interval $[l_1, l_t]$. In particular, let l_1 (respectively, l_t) be the *left* (respectively, the *right*) *element* of $N(G)$, denoted by $l(N(G))$ (respectively, by $r(N(G))$). We also define $N(G) = [0, 0]$ if G does not contain any crucial path partitions.

3. A sequential algorithm

3.1. Properties

In the following sections, let G be a distance-hereditary graph with twin set S, formed from two distance-hereditary graphs G_1 and G_2 with twin sets S_1 and S_2, respectively. Further, let $P_G[u, v]$ denote a path of G with end-vertices u and v. We allow the case of $u = v$, whereby $P_G[u, v]$ contains exactly one vertex. For two paths P_1 and P_2 containing exactly one common end-vertex, let $P_1 + P_2$ denote the concatenation of P_1 and P_2.

Let $Q = \{P_1, P_2, \ldots, P_l\}$ be a path partition of G. For convenience, we regard Q as a subgraph of G. Obviously, Q can be partitioned into three sets Q_1, Q_2 and Q_3, where Q_1 (respectively, Q_2) is the set of paths in Q that induces a subgraph in G_1 (respectively, G_2), and Q_3 is the set of the other paths in Q. For convenience, we sometimes represent Q as a triple (Q_1, Q_2, Q_3). For a distance-hereditary graph obtained by the true twin or attachment operation, the following definition provides a function which can construct a path partition from another path partition by adding some edges between S_1 and S_2.

Definition 2. Let $Q = (\{P_{G_1}[u_1, v_1], P_{G_1}[u_2, v_2], \ldots, P_{G_1}[u_i, v_i]\}, \{P_{G_2}[x_1, y_1], P_{G_2}[x_2, y_2], \ldots, P_{G_2}[x_j, y_j]\}, Q_3)$ be a path partition of $G = G_1 \otimes G_2$ or $G = G_1 \oplus G_2$. We first define a function from path partitions to themselves for $i \geqslant j$ as follows. Given an odd integer $k \leqslant 2j-1$, let $P_k = P_{G_1}[u_1, v_1] + (v_1, x_1) + P_{G_2}[x_1, y_1] + (y_1, u_2) + \cdots + (y_{\lceil \frac{k}{2} \rceil - 1}, u_{\lceil \frac{k}{2} \rceil}) + P_{G_1}[u_{\lceil \frac{k}{2} \rceil}, v_{\lceil \frac{k}{2} \rceil}] + (v_{\lceil \frac{k}{2} \rceil}, x_{\lceil \frac{k}{2} \rceil}) + P_{G_2}[x_{\lceil \frac{k}{2} \rceil}, y_{\lceil \frac{k}{2} \rceil}]$ (as shown in Fig. 2(a)); and given an even integer $k < 2j$ (respectively, $k \leqslant 2j$) for $i = j$ (respectively, $i \neq j$), let $P_k = P_{k-1} + (y_{\frac{k}{2}}, u_{\frac{k}{2}+1}) + P_{G_2}[u_{\frac{k}{2}+1}, v_{\frac{k}{2}+1}]$ (as shown in Fig. 2(b)). Define function $\Phi_k(Q_1, Q_2, Q_3) = (Q_1', Q_2', Q_3')$, where

$$Q_1' = \{P_{G_1}[u_{\frac{k}{2}+2}, v_{\frac{k}{2}+2}], P_{G_1}[u_{\frac{k}{2}+3}, v_{\frac{k}{2}+3}], \ldots, P_{G_1}[u_i, v_i]\},$$

$$Q_2' = \{P_{G_2}[x_{\frac{k}{2}+1}, y_{\frac{k}{2}+1}], P_{G_2}[x_{\frac{k}{2}+2}, y_{\frac{k}{2}+2}], \ldots, P_{G_2}[x_j, y_j]\},$$

and $Q_3' = Q_3 \cup P_k$ if k is even; and where

$$Q_1' = \{P_{G_1}[u_{\lceil \frac{k}{2} \rceil + 1}, v_{\lceil \frac{k}{2} \rceil + 1}], P_{G_1}[u_{\lceil \frac{k}{2} \rceil + 2}, v_{\lceil \frac{k}{2} \rceil + 2}], \ldots, P_{G_1}[u_i, v_i]\},$$

$$Q_2' = \{P_{G_2}[x_{\lceil \frac{k}{2} \rceil + 1}, y_{\lceil \frac{k}{2} \rceil + 1}], P_{G_2}[x_{\lceil \frac{k}{2} \rceil + 2}, y_{\lceil \frac{k}{2} \rceil + 2}], \ldots, P_{G_2}[x_j, y_j]\},$$

and $Q_3' = Q_3 \cup P_k$ if k is odd. On the other hand, for $i < j$, the definition is similar to that of $i \geqslant j$, which is obtained by reversing the roles of Q_1 and Q_2. In the special case of $Q_3 = \emptyset$, we simply denote $\Phi_k(Q_1, Q_2, Q_3)$ by $Q_1 \Phi_k Q_2$.

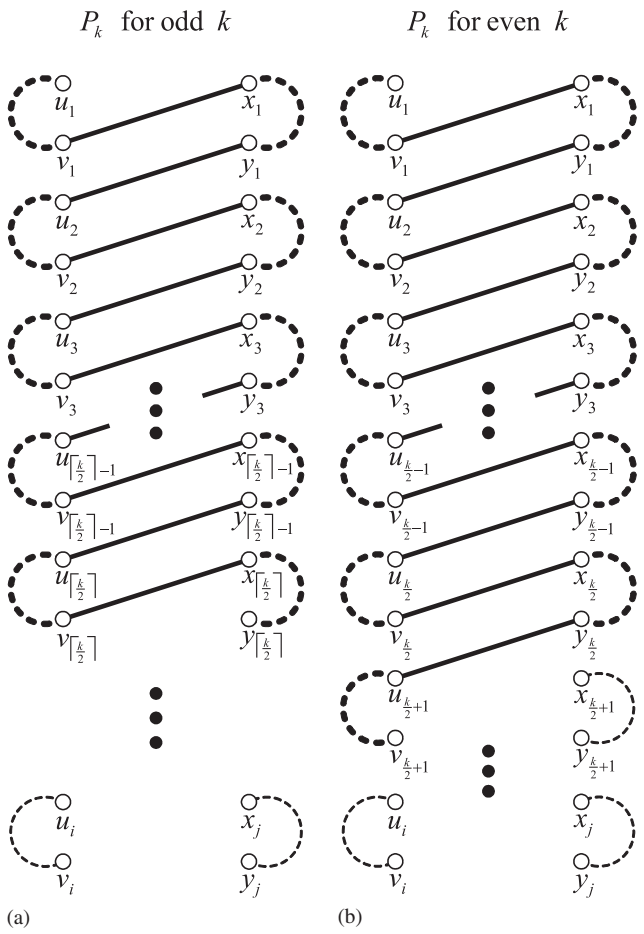

Fig. 2. An illustration of P_k in Definition 2.

Lemma 3. *Let $G = G_1 \otimes G_2$. A crucial q-path partition $\mathscr{P}^q(G)$ can be constructed from a crucial i-path partition $\mathscr{P}^i(G_1)$ and a crucial j-path partition $\mathscr{P}^j(G_2)$, where* $\textsc{Max}\{1, \textsc{Max}\{i, j\} - \textsc{Min}\{i, j\}\} \leqslant q \leqslant i + j$.

Proof. Without loss of generality, assume that $i \geqslant j$. The other case of $i < j$ can be shown similarly. When $i > j$ (respectively, $i = j$), $\mathscr{P}^q(G)$ for $i - j \leqslant q \leqslant i + j$ (respectively, $1 \leqslant q \leqslant 2j$), can be constructed as follows.

- $\mathscr{P}^{i+j}(G)$ can be obtained by the union of $\mathscr{P}^i(G_1)$ and $\mathscr{P}^j(G_2)$.
- $\mathscr{P}^{i+j-k}(G)$ for $1 \leqslant k \leqslant 2j$ (respectively, $1 \leqslant k \leqslant 2j - 1$), can be obtained by $\mathscr{P}^i(G_1)\Phi_k\mathscr{P}^j(G_2)$ if $i > j$ (respectively, $i = j$). □

Lemma 4. *Let $G = G_1 \oplus G_2$. A crucial path partition $\mathscr{P}^{i-j}(G)$ can be constructed from $\mathscr{P}^i(G_1)$ and $\mathscr{P}^j(G_2)$* if $i > j$.

Proof. By constructing $\mathscr{P}^i(G_1)\Phi_{2j}\mathscr{P}^j(G_2)$, the result holds. □

Consider an arbitrary crucial path partition $\mathscr{P}^q(G)$ in a connected distance-hereditary graph G formed from G_1 and G_2. Recall that we can regard a set of paths $\mathscr{P}^q(G)$ as a graph. If we *restrict* $\mathscr{P}^q(G)$ to G_1 (respectively, G_2) by considering the subgraph of $\mathscr{P}^q(G)$ induced by those vertices in G_1 (respectively, G_2), then a crucial path partition, denoted by $\mathscr{P}^i(G_1)$ (respectively, $\mathscr{P}^j(G_2)$), is obtained. For ease of reference, we call $\mathscr{P}^i(G_1)$ (respectively, $\mathscr{P}^j(G_2)$) the *projection of* $\mathscr{P}^q(G)$ *in* G_1 (respectively, G_2). For a Hamiltonian cycle C of G, the projections of C in G_1 and G_2 can be defined similarly. If we represent $\mathscr{P}^q(G)$ as a triple (Q_1, Q_2, Q_3), where Q_1 (respectively, Q_2) is the set

composed of those paths whose vertices only belong to V_1 (respectively, V_2) and Q_3 is the set composed of those paths travelling between V_1 and V_2, then $Q_1 \subseteq \mathscr{P}^i(G_1)$, $Q_2 \subseteq \mathscr{P}^j(G_2)$ and Q_3 is obtained by merging $\mathscr{P}^i(G_1) - Q_1$ and $\mathscr{P}^j(G_2) - Q_2$ with some edges in $S_1 \times S_2$. Note that $Q_2 = \emptyset$ when $G = G_1 \oplus G_2$. We call those edges in $S_1 \times S_2$ used to form Q_3 *marked edges of* $\mathscr{P}^q(G)$, denoted by $M(\mathscr{P}^q(G))$.

The number q, which equals $|Q_1| + |Q_2| + |Q_3|$, can be measured in terms of i, j and $|M(\mathscr{P}^q(G))|$. Consider the process to generate $\mathscr{P}^q(G)$ by adding the edges of $M(\mathscr{P}^q(G))$, one at a time, to the current path partition starting from $\mathscr{P}^i(G_1) \cup \mathscr{P}^j(G_2)$. The process is repeated until all the marked edges are considered. Observe that, after adding each edge, the number of paths in the resulting path partition is reduced by one. Therefore, $q = i + j - |M(\mathscr{P}^q(G))|$. On the other hand, for a path in $\mathscr{P}^i(G_1) \cup \mathscr{P}^j(G_2)$ with different endpoints (respectively, the same endpoint), each endpoint can be incident to at most one edge (respectively, two edges) in $M(\mathscr{P}^q(G))$ to form $\mathscr{P}^q(G)$. Thus $0 \leqslant |M(\mathscr{P}^q(G))| \leqslant \text{MIN}\{2i, 2j\}$ if $i \neq j$, and $0 \leqslant |M(\mathscr{P}^q(G))| \leqslant 2i - 1$ if $i = j$. In particular, when $G = G_1 \oplus G_2$, both the conditions $i > j$ and $|M(\mathscr{P}^q(G))| = 2j$ hold. In summary, we have the following lemma.

Lemma 5. *Assume that* $G = G_1 \otimes G_2$. *Let* $\mathscr{P}^q(G)$ *be an arbitrary crucial path partition of* G *and let* $\mathscr{P}^i(G_1)$ *and* $\mathscr{P}^j(G_2)$ *be its projections in* G_1 *and* G_2, *respectively. Then,* $q = i + j - |M(\mathscr{P}^q(G))|$, *where* $0 \leqslant |M(\mathscr{P}^q(G))| \leqslant \text{MIN}\{2i, 2j\}$ *if* $i \neq j$, *and* $0 \leqslant |M(\mathscr{P}^q(G))| \leqslant 2i - 1$ if $i = j$.

Lemma 6. *Assume that* $G = G_1 \oplus G_2$. *Let* $\mathscr{P}^q(G)$ *be an arbitrary crucial path partition of* G *and let* $\mathscr{P}^i(G_1)$ *and* $\mathscr{P}^j(G_2)$ *be its projections in* G_1 *and* G_2, *respectively. Then,* $i > j$ *and* $q = i - j$.

Lemma 7. *Assume that* $G = G_1 \otimes G_2$. *Let* a_i *and* b_i *be positive integers for* $i \in \{1, 2\}$. *If* $N(G_1) = [a_1, b_1]$ *and* $N(G_2) = [a_2, b_2]$, *then*

$$N(G) = \begin{cases} [1, b_1 + b_2] & \text{if } [a_1, b_1] \cap [a_2, b_2] \neq \emptyset; \\ [a_2 - b_1, b_1 + b_2] & \text{if } [a_1, b_1] \cap [a_2, b_2] = \emptyset \text{ and } a_2 > b_1; \\ [a_1 - b_2, b_1 + b_2] & \text{if } [a_1, b_1] \cap [a_2, b_2] = \emptyset \text{ and } a_1 > b_2. \end{cases}$$

Proof. By Lemma 3, a crucial path partition $\mathscr{P}^q(G)$ can be constructed from the two crucial path partitions $\mathscr{P}^i(G_1)$ and $\mathscr{P}^j(G_2)$ for $\text{MAX}\{1, \text{MAX}\{i, j\} - \text{MIN}\{i, j\}\} \leqslant q \leqslant i + j$. Since $a_1 \leqslant i \leqslant b_1$ and $a_2 \leqslant j \leqslant b_2$, all possible constructions specified in the statement can be obtained.

According to Lemma 5, $\mathscr{P}^q(G)$ with $q > b_1 + b_2$ does not exist in G by the following reasoning. If $i + j - |M(P^q(G))| > b_1 + b_2$, then $i + j > b_1 + b_2$. This leads to $i > b_1$ or $j > b_2$, which contradicts the assumption. Therefore, the right element of $N(G)$ equals $b_1 + b_2$.

We next show that the left elements specified in the statement are correct. If $[a_1, b_1] \cap [a_2, b_2] = \emptyset$ and $a_2 > b_1$, then, by Lemma 5, q has the minimum value $a_2 - b_1$. (This is explained below.) Assume that $q = i + j - |M(P^q(G))| < a_2 - b_1$ holds, where $i \in [a_1, b_1]$ and $j \in [a_2, b_2]$. Clearly, $i < j$ and $0 \leqslant |M(\mathscr{P}^q(G))| \leqslant 2i$. Then, $j - i = i + j - 2i \leqslant i + j - |M(\mathscr{P}^q(G))| < a_2 - b_1$, so $0 \leqslant j - a_2 < i - b_1 \leqslant 0$, is a contradiction. Therefore, $q \geqslant a_2 - b_1$ in this case. The case of $[a_1, b_1] \cap [a_2, b_2] = \emptyset$ and $a_1 > b_2$ can be argued similarly. □

The above lemma can be further simplified as follows.

Corollary 1. *Assume that* $G = G_1 \otimes G_2$. *Let* a_i *and* b_i *be positive integers for* $i \in \{1, 2\}$. *If* $N(G_1) = [a_1, b_1]$ *and* $N(G_2) = [a_2, b_2]$, *then* $N(G) = [\text{MAX}\{1, a_2 - b_1, a_1 - b_2\}, b_1 + b_2]$.

Lemma 8. *Assume that* $G = G_1 \oplus G_2$. *Let* a_i *and* b_i *be positive integers for* $i \in \{1, 2\}$. *If* $N(G_1) = [a_1, b_1]$ *and* $N(G_2) = [a_2, b_2]$, *then*

$$N(G) = \begin{cases} [0, 0] & \text{if } a_2 \geqslant b_1; \\ [1, b_1 - a_2] & \text{if } [a_1, b_1] \cap [a_2, b_2] \neq \emptyset \text{ and } a_2 < b_1; \\ [a_1 - b_2, b_1 - a_2] & \text{if } [a_1, b_1] \cap [a_2, b_2] = \emptyset \text{ and } a_2 < b_1. \end{cases}$$

Proof. From Lemmas 4 and 6, $q \in N(G)$ if and only if there exists $i \in N(G_1)$ and $j \in N(G_2)$ such that $i - j = q$. When $a_2 \geqslant b_1$, since there are no $i \in N(G_1)$ and $j \in N(G_2)$ such that $i > j$, $N(G) = [0, 0]$.

We then consider the situation where $a_2 < b_1$. By Lemma 4, $\mathscr{P}^{i-j}(G)$ can be constructed from $\mathscr{P}^i(G_1)$ and $\mathscr{P}^j(G_2)$ if $i > j$. Clearly, all possible constructions specified in the statement can be obtained. By Lemma 6, the two elements of $N(G)$ for each situation can be obtained. □

Corollary 2. *Assume that $G = G_1 \oplus G_2$. Let a_i and b_i be positive integers for $i \in \{1, 2\}$. If $N(G_1) = [a_1, b_1]$ and $N(G_2) = [a_2, b_2]$, then $N(G) = [\text{MIN}\{\text{MAX}\{0, b_1 - a_2\}, \text{MAX}\{1, a_1 - b_2\}\}, \text{MAX}\{0, b_1 - a_2\}]$.*

Lemma 9. *Assume that $G = G_1 \odot G_2$. Let a_i and b_i be positive integers for $i \in \{1, 2\}$. If $N(G_1) = [a_1, b_1]$ and $N(G_2) = [a_2, b_2]$, then $N(G) = [a_1 + a_2, b_1 + b_2]$.*

Proof. It follows from the structure characterization that no vertex of G_1 is adjacent to any vertex of G_2. □

Theorem 2. *Let T_G be a decomposition tree of a distance-hereditary graph G. If $N(G) \neq [0, 0]$, then $N(G_v) = [a_v, b_v]$ for all nodes v in T_G, where $a_v \leqslant b_v \in \mathbf{Z}^+$.*

Proof. Note that $N(G_v) \neq [0, 0]$ because $N(G) \neq [0, 0]$. The theorem can be proved by induction on the height of T_G, according to Lemmas 7–9. □

Theorem 3. *Let $G = G_1 \otimes G_2$ or $G = G_1 \oplus G_2$ and let $|V(G)| \geqslant 3$. Then, G has a Hamiltonian cycle if and only if $N(G_i) \neq [0, 0]$, $i \in \{1, 2\}$, and $N(G_1) \cap N(G_2) \neq \emptyset$.*

Proof. We first show the "if" part. Let $k \in N(G_1) \cap N(G_2)$. Further, let $\mathscr{P}^k(G_1) = \{P_{G_1}[u_1, v_1], P_{G_1}[u_2, v_2], \ldots, P_{G_1}[u_k, v_k]\}$; and let $\mathscr{P}^k(G_2) = \{P_{G_2}[x_1, y_1], P_{G_2}[x_2, y_2], \ldots, P_{G_2}[x_k, y_k]\}$. Since $G = G_1 \otimes G_2$ or $G = G_1 \oplus G_2$, $\mathscr{P}^k(G_1)\Phi_{2k-1}\mathscr{P}^k(G_2)$ along with (u_1, y_k) forms a Hamiltonian cycle of G. Since $|V(G)| \geqslant 3$, the case of $N(G_1) = N(G_2) = [1, 1]$, $u_1 = v_1$, and $x_1 = y_1$ is impossible.

We next show the "only if" part. Consider an arbitrary Hamiltonian cycle C of G. Let $\mathscr{P}^i(G_1)$ and $\mathscr{P}^j(G_2)$ be the projections of C in G_1 and G_2, respectively. Moreover, $i = j$ because the number of end-vertices in $\mathscr{P}^i(G_1)$ is the same as $\mathscr{P}^j(G_2)$. By Lemma 7 and Lemma 8, $i \in N(G_1)$ and $i \in N(G_2)$. Therefore, $i(=j) \in N(G_1) \cap N(G_2)$. □

3.2. A linear-time algorithm

In this section, we develop an $O(n + m)$-time sequential algorithm to solve the Hamiltonian problem on distance-hereditary graphs. Consider a Hamiltonian distance-hereditary graph G. Let v be an arbitrary non-root node of T_G. Given a Hamiltonian cycle C of G, the projection of this cycle in G_v is a crucial k-path partition of G_v for some $1 \leqslant k \leqslant |V(G_v)|$. In particular, we denote such a k as the *target number* $tar_C(v)$ *of* G_v (the subscript C can be omitted from the notation if no ambiguity arises). Note that $tar(l) = 1$ if l is a leaf of T_G. For a non-root internal node $v \in V(T_G)$ with two children u and w, $\mathscr{P}^{tar(v)}(G_v)$ can be constructed from $\mathscr{P}^{tar(u)}(G_u)$ and $\mathscr{P}^{tar(w)}(G_w)$ by augmenting the marked edges $M(\mathscr{P}^{tar(v)}(G_v)) \subset S_u \times S_w$.

On the other hand, assume that G is obtained from G_1 and G_2 with $N(G_1) = [a_1, b_1]$ and $N(G_2) = [a_2, b_2]$, respectively. Given $N(G)$ and a fixed non-zero number $q \in N(G)$, we summarize the values of q_1 and q_2 in Table 1 such that $\mathscr{P}^q(G)$ can be constructed from $\mathscr{P}^{q_1}(G_1)$ and $\mathscr{P}^{q_2}(G_2)$, as shown by the following example.

Example 1. Consider the $\otimes$ entry in the table which means that $G = G_1 \otimes G_2$. Given a fixed integer q, q_1 and q_2 can be obtained by checking Conditions 1–3. For example, if $[a_1, b_1] \cap [a_2, b_2] \neq \emptyset$ (classified by Condition 1), $b_1 = b_2$ (classified by Condition 2), and $1 \leqslant q \leqslant b_1 + b_2$ (classified by Condition 3), then $q_1 = b_1$ and $q_2 = b_2$, as shown in the first row of Table 1.

We now develop an $O(n + m)$-time sequential algorithm to solve the Hamiltonian problem on distance-hereditary graphs. Since a 2-vertex graph has no Hamiltonian cycle, we consider a connected graph with $n > 2$. Given a decomposition tree T_G as an input instance, the algorithm consists of the following three phases.

Table 1
Determining q_1 and q_2 so that $\mathscr{P}^q(G)$ can be constructed from $\mathscr{P}^{q_1}(G_1)$ and $\mathscr{P}^{q_2}(G_2)$

Type	Condition 1	Range of q	Condition 2	Condition 3	q_1	q_2
$\otimes$	$[a_1,b_1]\cap[a_2,b_2]\neq\emptyset$	$[1, b_1+b_2]$	$b_1=b_2$	$1\leqslant q\leqslant b_1+b_2$	b_1	b_2
			$b_1<b_2$	$1\leqslant q\leqslant b_1$	b_1	b_1
				$b_1+1\leqslant q\leqslant b_2$	b_1	q
			$b_1>b_2$	$b_2+1\leqslant q\leqslant b_1+b_2$	b_1	b_2
				$1\leqslant q\leqslant b_2$	b_2	b_2
				$b_2+1\leqslant q\leqslant b_1$	q	b_2
				$b_1+1\leqslant q\leqslant b_1+b_2$	b_1	b_2
	$[a_1,b_1]\cap[a_2,b_2]=\emptyset$	$[a_2-b_1, b_1+b_2]$	$a_2>b_1$	$a_2-b_1\leqslant q\leqslant b_2-b_1$	b_1	$q+b_1$
				$b_2-b_1+1\leqslant q\leqslant b_1+b_2$	b_1	b_2
		$[a_1-b_2, b_1+b_2]$	$a_1>b_2$	$a_1-b_2\leqslant q\leqslant b_1-b_2$	$q+b_2$	b_2
				$b_1-b_2+1\leqslant q\leqslant b_1+b_2$	b_1	b_2
$\oplus$	$[a_1,b_1]\cap[a_2,b_2]\neq\emptyset$	$[1, b_1-a_2]$	$a_2\leqslant a_1\leqslant b_2\leqslant b_1$ but $b_1>a_2$	$1\leqslant q\leqslant b_1-a_1$	a_1+q	a_1
				$b_1-a_1+1\leqslant q\leqslant b_1-a_2$	b_1	b_1-q
			$a_1\leqslant a_2<b_1\leqslant b_2$	$1\leqslant q\leqslant b_1-a_2$	a_2+q	a_2
		$[0,0]$	$a_2=b_1$	None		
	$[a_1,b_1]\cap[a_2,b_2]=\emptyset$	$[0,0]$	$a_2>b_1$	None		
		$[a_1-b_2, b_1-a_2]$	$a_2<b_1$	$a_1-b_2\leqslant q\leqslant b_1-b_2$	b_2+q	b_2
				$b_1-b_2+1\leqslant q\leqslant b_1-a_2$	b_1	b_1-q
$\odot$	$[a_1,b_1]\cap[a_2,b_2]\neq\emptyset$	$[a_1+a_2, b_1+b_2]$	$b_1=b_2$	$a_1+a_2\leqslant q\leqslant b_1+a_2$	$q-a_2$	a_2
				$b_1+a_2+1\leqslant q\leqslant b_1+b_2$	b_1	$q-b_1$
			$b_1<b_2$	$a_1+a_2\leqslant q\leqslant a_1+b_2$	a_1	$q-a_1$
				$a_1+b_2+1\leqslant q\leqslant b_1+b_2$	$q-b_2$	b_2
			$b_1>b_2$	$a_1+a_2\leqslant q\leqslant a_2+b_1$	$q-a_2$	a_2
				$a_2+b_1+1\leqslant q\leqslant b_1+b_2$	b_1	$q-b_1$
	$[a_1,b_1]\cap[a_2,b_2]=\emptyset$		$a_2>b_1$	$a_1+a_2\leqslant q\leqslant a_1+b_2$	a_1	$q-a_1$
				$a_1+b_2+1\leqslant q\leqslant b_1+b_2$	$q-b_2$	b_2
			$a_1>b_2$	$a_1+a_2\leqslant q\leqslant a_2+b_1$	$q-a_2$	a_2
				$a_2+b_1+1\leqslant q\leqslant b_1+b_2$	b_1	$q-b_1$

Phase 1: Compute $N(G_v)$ for each non-root node[3] v of T_G. Initially, let $N(G_l)=[1,1]$ for each leaf l. By Lemmas 7–9, the desired values can be computed by a bottom-up evaluation process. During the computation, if $N(G_v)=[0,0]$ for some non-root node v, we terminate the algorithm and exit. Otherwise, all non-root nodes of T_G are associated with the desired values. Clearly, this phase takes $O(n)$ time. Before proceeding to the next phase, we give the following lemma, which is useful for proving the correctness of the algorithm.

Lemma 10. *In Phase* 1, *if there is a non-root node* v *with* $N(G_v)=[0,0]$, *then* G *is not Hamiltonian*.

Proof. Let y and z be the two children of the root of T_G. If v in $T_G(y)$ (respectively, $T_G(z)$), then G_y (respectively, G_z) does not contain any crucial path partitions. According to Theorem 3, G is not Hamiltonian. □

Phase 2: Compute the target number $tar(v)$ for each non-root node v. Assume that y and z are the left and right children of $root(T_G)$, respectively. Select an arbitrary number $k\in N(G_y)\cap N(G_z)$ and let $tar(y)=tar(z)=k$. For each non-root internal node v with the two children u and w, select $tar(u)$ and $tar(w)$ such that $\mathscr{P}^{tar(v)}(G_v)$ can be constructed from $\mathscr{P}^{tar(u)}(G_u)$ and $\mathscr{P}^{tar(w)}(G_w)$. Using Table 1, $tar(u)$ and $tar(w)$ can be found in $O(1)$ time. Therefore, this phase can be implemented to run in $O(n)$ time by a top-down evaluation process.

Phase 3: Find a set of edges to form a Hamiltonian cycle of G. Define $tar(root(T_G))=1$. During a bottom-up traverse of T_G, we maintain a list $L_v=(\langle p_1,q_1\rangle,\langle p_2,q_2\rangle,\ldots,\langle p_{tar(v)},q_{tar(v)}\rangle)$ and $M(\mathscr{P}^{tar(v)}(G_v))$ for each node v, where $\langle p_i,q_i\rangle$ represents the end-vertices of $P_{G_v}[p_i,q_i]\in\mathscr{P}^{tar(v)}(G_v)$. Initially, for each leaf l representing a

[3] Note that we do not need to compute $N(G_{root(T_G)})$ and $tar(root(T_G))$ for solving the problem using our sequential algorithm. But, both values will be determined in our parallel algorithm in order to obtain the target numbers of all non-root nodes using the tree contraction technique.

primitive distance-hereditary graph $G_l = (\{v_l\}, \emptyset)$, let $L_l = (\langle v_l, v_l \rangle)$, and let $M(\mathscr{P}^1(G_l)) = \emptyset$. For each non-$\odot$ node v, $M(\mathscr{P}^{tar(v)}(G_v))$ can be found in $O(|M(\mathscr{P}^{tar(v)}(G_v))|)$ time, according to Definition 2 and the proofs of Lemmas 3 and 4. Assume that u and w be the two children of v. Similarly, L_v can be constructed from L_u and L_w in constant time by union of the two lists if v is a $\odot$-node, or in $O(|M(\mathscr{P}^{tar(v)}(G_v))|)$ time if v is a non-$\odot$-node. Since all the marked edges together with some edge form aHamiltonian cycle, this phase takes $O(n)$ time.

By Lemmas 7–10, and Theorem 3, it is clear that if the given graph is not Hamiltonian, our algorithm will detect it and then exit. Otherwise, a Hamiltonian cycle of G will be generated. By the above discussion and Lemma 1, we have the following result.

Theorem 4. *The Hamiltonian problem on distance-hereditary graphs can be solved in* $O(n + m)$ *sequential time.*

4. A parallel algorithm

In this section, we present a parallel algorithm to solve the Hamiltonian problem on distance-hereditary graphs. Given a decomposition tree T_G, our algorithm is presented in the following subsections.

4.1. Computing $N(G_v)$

Let u and w be the left and right children, respectively, of an internal node v in T_G. Also, let $N(G_u) = [a_1, b_1]$ and $N(G_w) = [a_2, b_2]$. If v is $\otimes$ (respectively, $\odot$), then the formulas in Corollary 1 (respectively, Lemma 9) are used to compute $N(G_v)$, after relaxing the original constraint requiring that two numbers a_i and b_i, $i \in \{1, 2\}$, need to be positive integers. Here, we allow a_i and b_i, $i \in \{1, 2\}$, to be integers. For ease of parallel implementation, we adopt the following formula to compute $N(G_v)$ for $\oplus$-node, instead of that in Corollary 2:

$$N(G_v) = [\text{MAX}\{1, a_1 - b_2\}, b_1 - a_2]. \tag{1}$$

Recall that the input graph G considered in this paper is connected with $n > 2$. In fact, the type of $root(T_G)$ (that is, $\otimes$-node or $\oplus$-node) does not effect the correctness of our algorithms. Hereafter, for convenience, we further assume that $root(T_G)$ is a $\otimes$-node. By a bottom-up evaluation process, the following result can be obtained.

Lemma 11. *Assume that* $N(G_u)$ *is computed for each non-leaf node* u *using Eq.* (1) *and the formulas described in Corollary* 1 *and Lemma* 9. *Let* y *and* z *be the two children of* $root(T_G)$. *Then, G is Hamiltonian if and only if the right element* $r(N(G_v)) > 0$ *for all nodes* v, *and* $N(G_y) \cap N(G_z) \neq \emptyset$.

Proof. The result follows directly from Theorem 3 and the fact $N(G_v) \neq [0, 0]$ iff $r(N(G_v)) > 0$ for all nodes v. □

In the rest of this section, we apply the binary tree contraction technique described in [1] to compute $N(G_v)$ using Eq. (1) and the formulas described in Corollary 1 and Lemma 9. This technique recursively applies the two operations, *prune* and *bypass*, to a given binary tree. $Prune(u)$ removes a leaf node u from the current tree; $bypass(v)$ removes a node v with exactly one child w, and lets the parent of v become the new parent of w. After executing $O(\log n)$ phases, the original tree is contracted to be a three-node tree [1]. Fig. 3 shows an application of the two procedures $prune(u)$ and $bypass(v)$.

Lemma 12 (*Abrahamson et al. [1]*). *If the prune and bypass operations can be performed by one processor in a constant time, then the binary tree contraction algorithm can 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 the input binary tree.*

During the execution of the binary tree contractions, we construct a pair of functions with special form described in Definition 3 such that when the *prune* and *bypass* operations are executed the special form is also maintained for each remaining node of the current tree, and $N(G_v)$ can finally be computed for each node v.

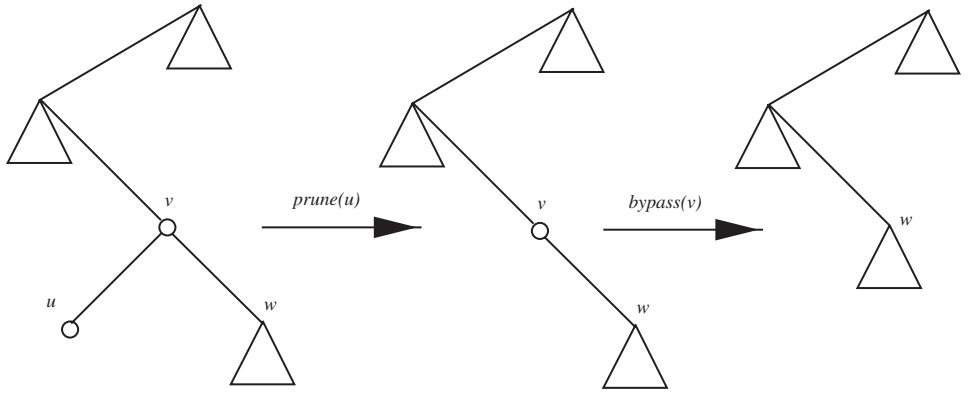

Fig. 3. Illustration of the two procedures $prune(u)$ and $bypass(v)$.

Definition 3. A pair of binary functions $F(x, y) = [f(x, y), g(x, y)]$, where $f, g : \mathbf{Z}^2 \mapsto \mathbf{Z}$, possess the *closed form* if the following two conditions hold. (1) $f(x, y) = \text{MAX}\{x + c_1, -y + c_2, d\}$, where $c_1, c_2, d \in \mathbf{Z} \cup \{-\infty\}$ and c_1, c_2, d cannot be $-\infty$ at the same time; and (2) $g(x, y) = e - \text{MAX}\{x + h_1, -y + h_2, j\}$, where $h_1, h_2, j \in \mathbf{Z} \cup \{-\infty\}$, $e \in \mathbf{Z}$, and h_1, h_2, j cannot be $-\infty$ at the same time.

For a pair of binary functions $F(x, y) = [f(x, y), g(x, y)]$, let $l(F)$ and $r(F)$ be $f(x, y)$ and $g(x, y)$, respectively.

Lemma 13. *Let F_1 and F_2 be two arbitrary function pairs possessing the closed form. Then, $F_1 \circ (l(F_2), r(F_2))$ also possess the closed form.*

Proof. Let $F_i(x, y) = [f_i(x, y), g_i(x, y)] = [\text{MAX}\{x + c_{i,1}, -y + c_{i,2}, d_i\}, e_i - \text{MAX}\{x + h_{i,1}, -y + h_{i,2}, j_i\}]$ for $i \in \{1, 2\}$. Clearly,

$$\begin{aligned}
l(F_1 \circ (l(F_2), r(F_2))) &= f_1 \circ (l(F_2), r(F_2)) \\
&= f_1 \circ (\text{MAX}\{x + c_{2,1}, -y + c_{2,2}, d_2\}, e_2 - \text{MAX}\{x + h_{2,1}, -y + h_{2,2}, j_2\}) \\
&= \text{MAX}\{\text{MAX}\{x + c_{2,1}, -y + c_{2,2}, d_2\} + c_{1,1}, -(e_2 - \text{MAX}\{x + h_{2,1}, \\
&\quad - y + h_{2,2}, j_2\}) + c_{1,2}, d_1\} \\
&= \text{MAX}\{\text{MAX}\{x + (c_{2,1} + c_{1,1}), -y + (c_{2,2} + c_{1,1}), d_2 + c_{1,1}\}, \\
&\quad c_{1,2} - e_2 + \text{MAX}\{x + h_{2,1}, -y + h_{2,2}, j_2\}, d_1\} \\
&= \text{MAX}\{\text{MAX}\{x + (c_{2,1} + c_{1,1}), -y + (c_{2,2} + c_{1,1}), d_2 + c_{1,1}\}, \\
&\quad \text{MAX}\{x + (c_{1,2} - e_2 + h_{2,1}), -y + (c_{1,2} - e_2 + h_{2,2}), c_{1,2} - e_2 + j_2\}, d_1\} \\
&= \text{MAX}\{x + \text{MAX}\{c_{2,1} + c_{1,1}, c_{1,2} - e_2 + h_{2,1}\}, -y + \text{MAX}\{c_{2,2} + c_{1,1}, \\
&\quad c_{1,2} - e_2 + h_{2,2}\}, \text{MAX}\{d_2 + c_{1,1}, c_{1,2} - e_2 + j_2, d_1\}\}. \qquad (2)
\end{aligned}$$

Therefore, $l(F_1 \circ (l(F_2), r(F_2)))$ possesses the closed form.

On the other hand,

$$\begin{aligned}
r(F_1 \circ (l(F_2), r(F_2))) &= g_1 \circ (l(F_2), r(F_2)) \\
&= g_1 \circ (\text{MAX}\{x + c_{2,1}, -y + c_{2,2}, d_2\}, e_2 - \text{MAX}\{x + h_{2,1}, -y + h_{2,2}, j_2\}) \\
&= e_1 - \text{MAX}\{\text{MAX}\{x + c_{2,1}, -y + c_{2,2}, d_2\} + h_{1,1}, -(e_2 - \text{MAX}\{x + h_{2,1}, \\
&\quad - y + h_{2,2}, j_2\}) + h_{1,2}, j_1\}. \qquad (3)
\end{aligned}$$

Using the method to derive Eq. (2), the term $\text{MAX}\{\text{MAX}\{x + c_{2,1}, -y + c_{2,2}, d_2\} + h_{1,1}, -(e_2 - \text{MAX}\{x + h_{2,1}, -y + h_{2,2}, j_2\}) + h_{1,2}, j_1\}$ of Eq. (3) can be simplified as $\text{MAX}\{x + \text{MAX}\{c_{2,1} + h_{1,1}, h_{1,2} - e_2 + h_{2,1}\}, -y + \text{MAX}\{c_{2,2} + h_{1,1}, h_{1,2} - e_2 + h_{2,2}\}, \text{MAX}\{d_2 + h_{1,1}, h_{1,2} - e_2 + j_2, j_1\}\}$. Therefore, $r(F_1 \circ (l(F_2), r(F_2)))$ possesses the closed form. □

Given a decomposition tree T_G, we next develop a parallel algorithm to compute $N(G_v)$ for each node v in T_G. For a node u in the current tree H, let $par_H(u)$ (respectively, $child_H(u)$) denote the *parent* (respectively, *children*) of u, and let $sib_H(u)$ denote the *sibling* of u. The subscript H can be omitted if no ambiguity arises. Recall that $l(N(G_u))$ (respectively, $r(N(G_u))$) is the *left* (respectively, *right*) *element* of the $N(G_u)$.

During the process of executing the tree contraction, we aim to construct a pair of binary functions $[f_v(x, y), g_v(x, y)]$ associated with each node v of the current tree, such that both functions possess the closed form and satisfy the invariant described below. Let v be an internal node in the current tree whose left and right children are u and w, respectively. Also let $\delta(u, v)$ and $\delta(w, v)$ be the left and right children of v in the original tree, respectively. Note that $\delta(u, v)$ and $\delta(w, v)$ are ancestors of u and w in the original tree, respectively. Hereafter, we call $\delta(u, v)$ (respectively, $\delta(w, v)$) *replacing ancestors of* u (respectively, w) *with respect to* v, and abbreviate it to $\delta(u)$ (respectively, $\delta(w)$) if no ambiguity arises.

INVARIANT: Once $l(N(G_u))$, $r(N(G_u))$, $l(N(G_w))$, and $r(N(G_w))$ are computed and provided as the inputs of $[f_u(x, y), g_u(x, y)]$ and $[f_w(x, y), g_w(x, y)]$, the following three statements hold:

S1: $N(G_{\delta(u)}) = [f_u(l(N(G_u)), r(N(G_u))), g_u(l(N(G_u)), r(N(G_u)))]$;

S2: $N(G_{\delta(w)}) = [f_w(l(N(G_w)), r(N(G_w))), g_w(l(N(G_w)), r(N(G_w)))]$;

S3: According to Corollary 1, Eq. (1) and Lemma 9, $N(G_v)$ can be computed using $N(G_{\delta(u)})$ and $N(G_{\delta(w)})$.

For a node v in the current tree, we call the above functions $f_v(x, y)$ and $g_v(x, y)$ *the crucial functions of* v. We next describe the details of our algorithm. Initially, for each node v in the given tree we construct $f_v(x, y) = \text{MAX}\{x + 0, -y + (-\infty), -\infty\}$ and $g_v(x, y) = 0 - \text{MAX}\{x + (-\infty), -y + 0, -\infty\}$. Moreover, if v is a leaf, then let $l(N(G_v)) = r(N(G_v)) = 1$.

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. Assume that $[f_u(x, y), g_u(x, y)]$ and $[f_w(x, y), g_w(x, y)]$ are crucial functions of u and w in the current tree, respectively. Thus we have $N(G_{\delta(u)}) = [f_u(l(N(G_u)), r(N(G_u))), g_u(l(N(G_u)), r(N(G_u)))]$; $N(G_{\delta(w)}) = [f_w(l(N(G_w)), r(N(G_w))), g_w(l(N(G_w)), r(N(G_w)))]$. Since u is a leaf, $l(N(G_u)) = r(N(G_u)) = 1$. Therefore, $N(G_{\delta(u)})$ can be obtained through the function evaluation. On the other hand, since w may not be a leaf in the current tree, $l(N(G_w))$ and $r(N(G_w))$ are indeterminate values represented by the variables x and y, respectively. Hence, $N(G_{\delta(w)})$ can be represented by the interval $[f_w(x, y), g_w(x, y)]$. Let $s = f_u(1, 1)$; $t = g_u(1, 1)$; $l(N(G_{\delta(w)})) = f_w(x, y) = \text{MAX}\{x + c_1, -y + c_2, d\}$; $r(N(G_{\delta(w)})) = g_w(x, y) = e - \text{MAX}\{x + h_1, -y + h_2, j\}$. We then construct the following two intermediate functions in order to form $N(G_v)$ from those of $\delta(u)$ and $\delta(w)$:

Case A: v is a $\otimes$-node. According to Corollary 1, we construct the following functions.

Case A1: w is the left child of v. Then,

$$
\begin{aligned}
N(G_v) &= [\text{MAX}\{1, l(N(G_{\delta(u)})) - r(N(G_{\delta(w)})), l(N(G_{\delta(w)})) - r(N(G_{\delta(u)}))\}, \\
&\quad r(N(G_{\delta(w)})) + r(N(G_{\delta(u)}))] \\
&= [\text{MAX}\{1, s - g_w(x, y), f_w(x, y) - t\}, g_w(x, y) + t] \\
&= [\text{MAX}\{1, s - (e - \text{MAX}\{x + h_1, -y + h_2, j\}), \\
&\quad \text{MAX}\{x + c_1, -y + c_2, d\} - t\}, \\
&\quad (e - \text{MAX}\{x + h_1, -y + h_2, j\}) + t] \\
&= [\text{MAX}\{1, s - e + \text{MAX}\{x + h_1, -y + h_2, j\}, \\
&\quad \text{MAX}\{x + (c_1 - t), -y + (c_2 - t), \\
&\quad d - t\}\}, e + t - \text{MAX}\{x + h_1, -y + h_2, j\}] \\
&= [\text{MAX}\{1, \text{MAX}\{x + (s - e + h_1), -y + (s - e + h_2), s - e + j\}, \\
&\quad \text{MAX}\{x + (c_1 - t), -y + (c_2 - t), d - t\}\}, \\
&\quad (e + t) - \text{MAX}\{x + h_1, -y + h_2, j\}] \\
&= [\text{MAX}\{x + \text{MAX}\{s - e + h_1, c_1 - t\}, -y + \text{MAX}\{s - e + h_2, c_2 - t\}, \\
&\quad \text{MAX}\{1, s - e + j, d - t\}\}, (e + t) - \text{MAX}\{x + h_1, -y + h_2, j\}]. \qquad (4)
\end{aligned}
$$

Case A2: w is the right child of v_i. Then, $N(G_{v_i}) = [\text{MAX}\{1, l(N(G_{\delta(w)})) - r(N(G_{\delta(u)})), l(N(G_{\delta(u)})) - r(N(G_{\delta(w)}))\}, r(N(G_{\delta(u)})) + r(N(G_{\delta(w)}))] = [\text{MAX}\{1, f_w(x, y) - t, s - g_w(x, y)\}, bg_w(x, y) + t]$. The derivation of this case is similar to that of Case A1.

Case B: v is a $\oplus$-node. According to Eq. (1), we construct the following functions.

Case B1: w is the left child of v. Then, $N(G_v)=[\text{MAX}\{1, l(N(G_{\delta(w)}))-r(N(G_{\delta(u)}))\}, r(N(G_{\delta(w)}))-l(N(G_{\delta(u)}))]=[\text{MAX}\{x+(c_1-t), -y+(c_2-t), \text{MAX}\{1, d-t\}\}, (e-s)-\text{MAX}\{x+h_1, -y+h_2, j\}]$.

Case B2: w is the right child of v_i. Similarly, $N(G_v)=[\text{MAX}\{1, s-g_w(x,y)\}, t-f_w(x,y)]=[\text{MAX}\{x+(s-e+h_1), -y+(s-e+h_2), \text{MAX}\{1, s-e+j\}\}, t-\text{MAX}\{x+c_1, -y+c_2, d\}]$.

Case C: v is a $\odot$-node. According to Lemma 9, $N(G_v)=[l(N(G_{\delta(w)}))+l(N(G_{\delta(u)})), r(N(G_{\delta(w)}))+r(N(G_{\delta(u)}))]=[f_w(x,y)+s, g_w(x,y)+t]=[\text{MAX}\{x+(c_1+s), -y+(c_2+s), d+s\}, (e+t)-\text{MAX}\{x+h_1, -y+h_2, j\}]$.

Therefore, the functions representing $N(G_v)$ in Cases A–C all possess the closed form. Let H denote the current tree. We construct the above functions after executing $prune(u)$. Given the two functions $l(N(G_v))$ and $r(N(G_v))$ constructed above, the contribution to the left and right elements of $N(G_{par_H(v)})$ can be obtained using $f_v(l(N(G_v)), r(N(G_v)))$ and $g_v(l(N(G_v)), r(N(G_v)))$. These functions are constructed for w after executing $bypass(par_H(u))=bypass(v)$. By Lemma 13, the above functions possess the closed form. Therefore, during the process of executing the binary tree contractions, the crucial functions constructed after executing $prune(u)$ and $bypass(par(u))$ can be implemented in O(1) time using one processor.

According to Lemma 12 and the above method together with the unwrapping technique described in Section 4.2.2, we can compute $N(G_v)$ for all internal nodes v. Therefore, we have the following lemma.

Lemma 14. *The interval $N(G_v)$ for each node $v \in V(T_G)$, can be computed in* $O(\log n)$ *time using* $O(n/\log n)$ *processors on an EREW PRAM.*

After computing $N(G_v)$, we check whether G is Hamiltonian by Lemma 11. This can be implemented in O(1) time using $O(n)$ processors on an EREW PRAM. By Brent's scheduling principle [18], it can be simulated in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM[4] . If G is Hamiltonian, then a Hamiltonian cycle of G can be further generated the method described in the following sections.

4.2. Computing $tar(v)$

Given a decomposition tree T_G associated with $N(G_v)$, $v \in V(T_G)$, we present an algorithm to compute $tar(v)$ in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM based on the binary tree contraction technique.

For an internal node v with the two children u and w in T_G, recall that G_v is constructed from G_u and G_w using one of the three operations defined in Theorem 1. For a positive integer $q_v \in N(G_v)$, we call $q_u \in N(G_u)$ and $q_w \in N(G_w)$ *the contributing numbers of* q_v if a crucial q_v-path partition $\mathscr{P}^{q_v}(G_v)$ can be constructed from the crucial path partitions $\mathscr{P}^{q_u}(G_u)$ and $\mathscr{P}^{q_w}(G_w)$. Two functions, $f_u : N(G_v) \mapsto N(G_u)$ and $f_w : N(G_v) \mapsto N(G_w)$, are said to be *the contributing functions of* v if $f_u(q_v)=q_u$, and $f_w(q_v)=q_w$.

On the other hand, consider a distance-hereditary graph G obtained from the two distance-hereditary graphs G_1 and G_2. Recall that, for a fixed $q \in N(G)$, Table 1 returns the contributing numbers $q_1 \in N(G_1)$ and $q_2 \in N(G_2)$ of q. For ease of parallel implementation, Table 1 can be rewritten such that given $q \in N(G)$, $N(G_1)=[a_1, b_1]$ and $N(G_2)=[a_2, b_2]$, the values of q_1 (respectively, q_2) can be obtained using the closed formula. The results are shown in Table 2 (see Appendix).

Observation 1.

1. $\text{MIN}\{\text{MAX}\{\pm x+c, a\}, b\}=\begin{cases} b & \text{if } a \geqslant b; \\ \text{MAX}\{\text{MIN}\{\pm x+c, b\}, a\} & \text{if } a < b. \end{cases}$

2. $\text{MAX}\{\text{MIN}\{\pm x+c, a\}, b\}=\begin{cases} b & \text{if } a \leqslant b; \\ \text{MIN}\{\text{MAX}\{\pm x+c, b\}, a\} & \text{if } a > b. \end{cases}$

We call the form (1) *min–max form*, and the function with the min–max form *min–max function*. Note that the constant function possesses the min–max form by Observation 1, and the function $\text{MIN}\{x+d, e\}$ (respectively, $\text{MAX}\{x+d, e\}$) possesses the min–max form $\text{MIN}\{\text{MAX}\{x+d, -\infty\}, e\}$ (respectively, $\text{MIN}\{\text{MAX}\{x+d, e\}, \infty\}$). From the above, it

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

is clear that each entry q_i, $i \in \{1, 2\}$, of Table 2 is a min–max function with the variable q. Therefore, we have the following observation.

Observation 2. *For each non-root internal node $v \in T_G$ with the two children u and w, the contributing functions of v have the unique form* $\text{MIN}\{\text{MAX}\{\pm x + c, a\}, b\}$, *where x is a variable drawn from* $\mathbf{Z}^+$, $a \in \mathbf{Z} \cup \{-\infty\}$, $b \in \mathbf{Z} \cup \{\infty\}$, *and* $c \in \mathbf{Z}$.

Observation 3. *Let y be the term representing $+x$ or $-x$ for a variable x drawn from* $\mathbf{Z}^+$. *Then*,

1. $-\text{MAX}\{y + c, a\} = \text{MIN}\{-y - c, -a\}$;
2. $-\text{MIN}\{y + c, a\} = \text{MAX}\{-y - c, -a\}$.

Definition 4. A unary function class $\mathscr{H}$ is *closed under composition* if for two arbitrary functions $h_1, h_2 \in \mathscr{H}$, $h_1 \circ h_2 \in \mathscr{H}$.

Lemma 15. *The class of min–max functions is closed under composition.*

Proof. Let $f(x)$ and $g(x)$ be two arbitrary min–max functions. We show in the following that $f \circ g$ also possesses the min–max form.

Case 1: $f(x) = \text{MIN}\{\text{MAX}\{\pm x + c, a\}, b\}$ and $g(x) = \text{MIN}\{\text{MAX}\{\pm x + d, e\}, j\}$. Without loss of generality, assume that $f(x) = \text{MIN}\{\text{MAX}\{x + c, a\}, b\}$ and $g(x) = \text{MIN}\{\text{MAX}\{-x + d, e\}, j\}$. According to Observations 1–3,, the other situations can be shown similarly. Clearly,

$$f \circ g(x) = \text{MIN}\{\text{MAX}\{\text{MIN}\{\text{MAX}\{-x + d, e\}, j\} + c, a\}, b\}. \tag{5}$$

Case 1.1: $e \geqslant j$. Eq. (5) can be further simplified as a constant $\text{MIN}\{\text{MAX}\{j + c, a\}, b\}$.

Case 1.2: $e < j$. We have:

$$\begin{aligned} f \circ g(x) &= \text{MIN}\{\text{MAX}\{\text{MIN}\{\text{MAX}\{-x + d, e\}, j\} + c, a\}, b\} \\ &= \text{MIN}\{\text{MAX}\{\text{MAX}\{\text{MIN}\{-x + d, j\}, e\} + c, a\}, b\} / * \text{ by Observation 1 } * / \\ &= \text{MIN}\{\text{MAX}\{\text{MAX}\{\text{MIN}\{-x + (d + c), j + c\}, e + c\}, a\}, b\} \\ &= \text{MIN}\{\text{MAX}\{\text{MIN}\{-x + (d + c), j + c\}, \text{MAX}\{e + c, a\}\}, b\}. \end{aligned} \tag{6}$$

Case 1.2.1: $(j + c) \leqslant \text{MAX}\{e + c, a\}$. Then, according to Observation 1, Eq. (6) can be further simplified to a constant $\text{MIN}\{\text{MAX}\{e + c, a\}, b\}$.

Case 1.2.2: $(j + c) > \text{MAX}\{e + c, a\}$. Then, from Observation 1, $f \circ g(x)$ can be further simplified as $\text{MIN}\{\text{MIN}\{\text{MAX}\{-x + (d + c), \text{MAX}\{e + c, a\}\}, j + c\}, b\} = \text{MIN}\{\text{MAX}\{-x + (d + c), \text{MAX}\{e + c, a\}\}, \text{MIN}\{j + c, b\}\}$.

Case 2: $f(x) = \text{MAX}\{\text{MIN}\{\pm x + c, a\}, b\}$ and $g(x) = \text{MAX}\{\text{MIN}\{\pm x + d, e\}, j\}$. The proof is similar to that of Case 1.

Case 3: One function is $\text{MIN}\{\text{MAX}\{\pm x + c, a\}, b\}$ and the other is $\text{MAX}\{\text{MIN}\{\pm x + d, e\}, j\}$. Without loss of generality, assume that $f(x) = \text{MIN}\{\text{MAX}\{x + c, a\}, b\}$, and $g(x) = \text{MAX}\{\text{MIN}\{-x + d, e\}, j\}$. Then,

$$\begin{aligned} f \circ g(x) &= \text{MIN}\{\text{MAX}\{\text{MAX}\{\text{MIN}\{-x + d, e\}, j\} + c, a\}, b\} \\ &= \text{MIN}\{\text{MAX}\{\text{MAX}\{\text{MIN}\{-x + (d + c), e + c\}, j + c\}, a\}, b\} \\ &= \text{MIN}\{\text{MAX}\{\text{MIN}\{-x + (d + c), e + c\}, \text{MAX}\{j + c, a\}\}, b\}. \end{aligned} \tag{7}$$

Then, a simplification of the desired form can be made in a similar way to Cases 1.2.1 and 1.2.2.

According to the above cases, the result holds. □

In the rest of this section, we assume that the input of our algorithm is a decomposition tree T_G satisfying the following condition. For each non-root internal node v with two children u and w, the edges (u, v) and (w, v) are associated with the contributing functions f_u and f_w of v, respectively. In particular, two edges incident to $root(T_G)$ are associated with two identity functions. Note that the identity function clearly possesses the min–max form. Our parallel algorithm consists of two stages, namely the *tree contraction stage* and the *unwrapping stage*. In the former,

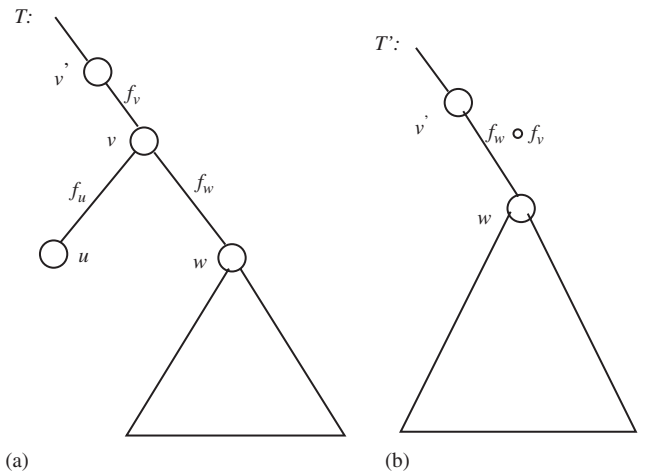

Fig. 4. An example of constructing min–max functions when $prune(u)$ and $bypass(par(u))$ are executed.

we use the binary tree contraction technique to contract the given tree into a tree-node tree T_3. During the contraction, we also construct min–max functions (described later) associated with the remaining edges of the current tree. In the latter, we restore T_3 into the original tree to compute the target numbers progressively.

4.2.1. The tree contraction stage

In the execution of tree contraction, assume that $prune(u)$ and $bypass(par(u))$ are performed consecutively. Without loss of generality, assume that u is the left child of $par(u) = v$ (the case of u being the right child can be handled similarly). Let $sib(u) = w$, and v' be the parent of v in the current tree (see Fig. 4(a)). Also assume that f_v, f_u and f_w are three contributing min–max functions associated with (v, v'), (u, v) and (w, v), respectively. After executing $prune(u)$ and then $bypass(v)$, the edge (w, v') is associated with the function $f_w \circ f_v$ (see Fig. 4(b)). Note that $f_w \circ f_v$ also possesses the min–max form by Lemma 15.

After executing the binary tree contractions, a three-node tree T_3 is obtained. Assume that y and z are the left and right children of T_G. Further, let $N(G_y) = [a_y, b_y]$ and $N(G_z) = [a_z, b_z]$. We first set $tar(root(T_G)) = tar(root(T_3)) = q \in [a_y, b_y] \cap [a_z, b_z]$, and then go to the next stage.

4.2.2. The unwrapping stage

This stage restores T_3 into T_G together with some function evaluations. We define two operations *arcprune* and *arcbypass*, denoted by $prune^{-1}$ and $bypass^{-1}$, respectively. $Prune^{-1}$ (respectively, $bypass^{-1}$) is the operation that restores the node deleted by *prune* (respectively, *bypass*), that is, $prune^{-1}(prune(u))=u$ (respectively, $bypass^{-1}(bypass(v))=v$). Consider an internal node w of the current tree from which two nodes u and v will be restored as follows (also see Fig. 5). Let $T(w)$ be the subtree rooted at w in the current tree T and $par_T(w) = t$. Without loss of generality, assume that $bypass^{-1}(bypass(v)) = v$, and $prune^{-1}(prune(u)) = u$ are restored consecutively such that in the current tree T', u is the left child of v and w is the other child of v in T'. Also assume that g_v and g_u are two min–max functions associated with (v, t) and (u, v), respectively. Note that $tar(t)$ was obtained in the previous step. We evaluate $tar(v) = g_v(tar(t))$ and let $tar(u) = 1$ because g_u must be the constant function 1 (see Fig. 5(b)).

After obtaining T_G, all the desired values are obtained. Since both composition (performed after executing *prune* and *bypass*) and evaluation (performed after executing *arcprune* and *arcbypass*) can be implemented in O(1) time using one processor, each stage takes $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM according to Lemma 12. Therefore, we have the following result.

Lemma 16. *Given a decomposition tree T_G, the target numbers for all nodes can be computed in* $O(\log n)$ *time using* $O(n/\log n)$ *processors on an EREW PRAM.*

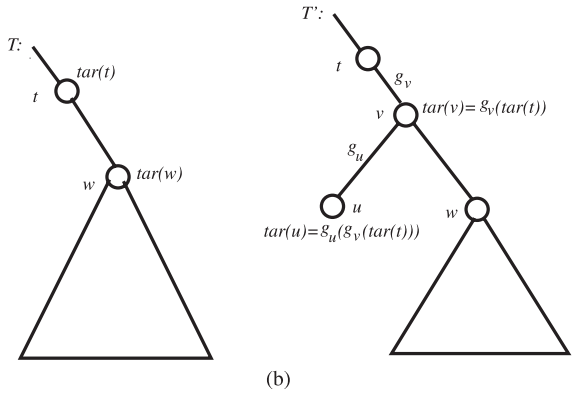

Fig. 5. An example of computing target numbers when *arcprune* and *arcbypass* are executed.

4.2.3. Complexities of our parallel algorithm

Since Phase 3 of our sequential algorithm can be implemented using the standard parallel techniques [17], such as the Euler-tour technique, list-marking technique and the pointer-jumping technique, we have the following lemma.

Lemma 17. *Given a decomposition tree T_G associated with $tar(v)$ for all $v \in V(T_G)$, a Hamiltonian cycle can be determined in* $O(\log n)$ *time using* $O(m/\log n)$ *processors on an EREW PRAM.*

We now summarize the result of the above discussion.

Theorem 5. *Given a decomposition tree of a distance-hereditary graph, the Hamiltonian problem can be solved in* $O(\log n)$ *time using* $O((n+m)/\log n)$ *processors on an EREW PRAM.*

By Lemma 2, the following corollary can be obtained immediately.

Corollary 3. *The Hamiltonian problem on distance-hereditary graphs can be solved in* $O(\log^2 n)$ *time using* $O(n+m)$ *processors on a CREW PRAM.*

5. Concluding remarks

In this paper, we improve the sequential time complexity of the Hamiltonian problem from $O(n^2)$ to $O(n+m)$. We also show that the Hamiltonian problem on distance-hereditary graphs can be efficiently solved in parallel. That is, the problem belongs to the NC class.

In [19], Miller and Teng presented a systematic 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 binary functions. We also show that a class of algebraic computation trees over $\{\mathbf{Z} \cup \{\pm\infty\}, \text{MIN}, \text{MAX}, +\}$ can be optimally evaluated using a class of functions that is closed under the function composition. In our future work, we will extend our technique to other graphs that are tree-representable.

Acknowledgements

We thank the anonymous referees for their careful review and very constructive suggestions and comments which helped to improve the content of this paper.

Appendix

Let G be a distance-hereditary graph constructed from the two distance-hereditary graphs G_1 and G_2. Assume that $N(G_1) = [a_1, b_1]$ and $N(G_2) = [a_2, b_2]$. The following Table 2 is refined from Table 1 such that the contributing numbers, q_1 and q_2, of $q \in N(G)$ can be obtained using the formulas in terms of q, a_i and b_i, $i \in \{1, 2\}$.

Table 2
A simplified version of Table 1

Type	Condition 1	Range of q	Condition 2	q_1	q_2
$\otimes$	$[a_1,b_1]\cap[a_2,b_2]\neq\emptyset$	$[1,b_1+b_2]$	$b_1=b_2$	b_1	b_2
			$b_1<b_2$	b_1	$\text{MIN}\{\text{MAX}\{q,b_1\},b_2\}$
			$b_1>b_2$	$\text{MIN}\{\text{MAX}\{q,b_2\},b_1\}$	b_2
	$[a_1,b_1]\cap[a_2,b_2]=\emptyset$	$[a_2-b_1,b_1+b_2]$	$a_2>b_1$	b_1	$\text{MIN}\{q+b_1,b_2\}$
		$[a_1-b_2,b_1+b_2]$	$a_1>b_2$	$\text{MIN}\{q+b_2,b_1\}$	b_2
$\oplus$	$[a_1,b_1]\cap[a_2,b_2]\neq\emptyset$	$[1,b_1-a_2]$	$a_2\leqslant a_1\leqslant b_2\leqslant b_1$ but $b_1>a_2$	$\text{MIN}\{q+a_1,b_1\}$	$\text{MIN}\{b_1-q,a_1\}$
			$a_1\leqslant a_2<b_1\leqslant b_2$	$q+a_2$	a_2
		$[0,0]$	$a_2=b_1$	None	
	$[a_1,b_1]\cap[a_2,b_2]=\emptyset$	$[0,0]$	$a_2>b_1$	None	
		$[a_1-b_2,b_1-a_2]$	$a_2<b_1$	$\text{MIN}\{q+b_2,b_1\}$	$\text{MIN}\{b_1-q,b_2\}$
$\odot$	$[a_1,b_1]\cap[a_2,b_2]\neq\emptyset$	$[a_1+a_2,b_1+b_2]$	$b_1=b_2$	$\text{MIN}\{q-a_2,b_1\}$	$\text{MAX}\{q-b_1,a_2\}$
			$b_1<b_2$	$\text{MAX}\{q-b_2,a_1\}$	$\text{MIN}\{q-a_1,b_2\}$
			$b_1>b_2$	$\text{MIN}\{q-a_2,b_1\}$	$\text{MAX}\{q-b_1,a_2\}$
	$[a_1,b_1]\cap[a_2,b_2]=\emptyset$		$a_2>b_1$	$\text{MAX}\{q-b_2,a_1\}$	$\text{MIN}\{q-a_1,b_2\}$
			$a_1>b_2$	$\text{MIN}\{q-a_2,b_1\}$	$\text{MAX}\{q-b_1,a_2\}$

References

[1] K. Abrahamson, N. Dadoun, D.G. Kirkpatrick, T. Przytycka, A simple parallel tree contraction algorithm, J. Algorithms 10 (1989) 287–302.
[2] H.J. Bandelt, H.M. Mulder, Distance-hereditary graphs, J. Combin. Theory Ser. B 41 (1) (1989) 182–208.
[3] C. Berge, Graphs and Hypergraphs, North-Holland, Amsterdam, 1973.
[4] A. Brandstädt, F.F. Dragan, A linear-time algorithm for connected γ-domination and Steiner tree on distance-hereditary graphs, Networks 31 (1998) 177–182.
[5] M.S. Chang, S.Y. Hsieh, G.H. Chen, Dynamic programming on distance-hereditary graphs, Proceedings of Seventh International Symposium on Algorithms and Computation (ISAAC'97), Lecture Notes in Computer Science, vol. 1350, 1997, pp. 344–353.
[6] E. Dahlhaus, Efficient parallel recognition algorithms of cographs and distance-hereditary graphs, Discrete Appl. Math. 57 (1) (1995) 29–44.
[7] A. D'atri, M. Moscarini, Distance-hereditary graphs, steiner trees, and connected domination, SIAM J. Comput. 17 (3) (1988) 521–538.
[8] F.F. Dragan, Dominating cliques in distance-hereditary graphs, Algorithm Theory—SWAT'94—Fourth Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science, vol. 824, Springer, Berlin, 1994, pp. 370–381.
[9] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic press, New York, 1980.
[10] P.L. Hammer, F. Maffray, Complete separable graphs, Discrete Appl. Math. 27 (1) (1990) 85–99.
[11] E. Howorka, A characterization of distance-hereditary graphs, Quart. J. Math. (Oxford) 28 (2) (1977) 417–420.
[12] S.Y. Hsieh, Parallel decomposition of distance-hereditary graphs, Proceedings of the Fourth 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 Computer Science, vol. 1557, 1999, pp. 417–426.
[13] S.Y. Hsieh, C.W. Ho, T.-S. Hsu, M.T. Ko, G.H. Chen, Efficient parallel algorithms on distance-hereditary graphs, Parallel Process. Lett. 9 (1) (1999) 43–52.
[14] S.Y. Hsieh, C.W. Ho, T.-S. Hsu, M.T. Ko, G.H. Chen, A faster implementation of a parallel tree contraction scheme and its application on distance-hereditary graphs, J. Algorithms 35 (2000) 50–81.
[15] S.Y. Hsieh, C.W. Ho, T.-S. Hsu, M.T. Ko, G.H. Chen, Characterization of efficient parallel solvable problems on distance-hereditary graphs, SIAM J. Discrete Math. 15 (4) (2002) 488–518.
[16] R.W. Hung, S.C. Wu, M.S. Chang, Hamiltonian cycle problem on distance-hereditary graphs, J. Inform. Sci. Engrg. 19 (2003) 827–838.
[17] J. JáJá, An Introduction to Parallel Algorithms, Addison Wesley, Reading, MA, 1992.
[18] R.M. Karp, V. Ramachandran, Parallel algorithms for shared memory machines, Handbook of Theoretical Computer Science, North-Holland, Amsterdam, 1990, pp. 869–941.
[19] G.L. Miller, S.H. Teng, Tree-based parallel algorithm design, Algorithmica 19 (1997) 369–389.
[20] H. Müller, F. Nicolai, Polynomial time algorithms for Hamiltonian problems on bipartite distance-hereditary graphs, Inform. Process. Lett. 46 (1993) 225–230.
[21] K. Nakano, S. Olariu, A.Y. Zomaya, A time-optimal solution form the path cover problems of cographs, Theoret. Comput. Sci. 290 (2003) 1541–1556.
[22] F. Nicolai, Hamiltonian problems on distance-hereditary graphs, Technique report, Gerhard-Mercator University, Germany, 1994.
[23] S.D. Nikolopoulos, Parallel algorithms for Hamiltonian problems on quasi-threshold graphs, J. Parallel Distributed Comput. 64 (2004) 48–67.
[24] S.D. Nikolopoulos, L. Palios, Efficient parallel recognition of cographs, Technical Report TR-2003-11 (http://charon.cs.uoi.gr/~tech_report/public.php), Department of Computer Science, University of Ioannina, 2003.
[25] H.G. Yeh, G.J. Chang, Weighted connected domination and Steiner trees in distance-hereditary graphs, Discrete Appl. Math. 87 (1–3) (1998) 245–253.

[26] H.G. Yeh, G.J. Chang, Linear-time algorithms for bipartite distance-hereditary graphs, Manuscript.
[27] H.G. Yeh, G.J. Chang, The path-partition problem in bipartite distance-hereditary graphs, Taiwaness J. Math. 2 (3) (1998) 353–360.
[28] H.G. Yeh, G.J. Chang, Weighted connected k-domination and weighted k-dominating clique in distance-hereditary graphs, Theoret. Comput. Sci. 263 (2001) 3–8.