Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

J. Parallel Distrib. Comput. 64 (2004) 48–67

Journal of Parallel and Distributed Computing

http://www.elsevier.com/locate/jpdc

Parallel algorithms for Hamiltonian problems on quasi-threshold graphs

Stavros D. Nikolopoulos

Department of Computer Science, University of Ioannina, P.O. Box 1186, GR-45110 Ioannina, Greece

Received 21 September 2001; revised 20 February 2003

Abstract

In this paper we show structural and algorithmic properties on the class of quasi-threshold graphs, or QT-graphs for short, and prove necessary and sufficient conditions for a QT-graph to be Hamiltonian. Based on these properties and conditions, we construct an efficient parallel algorithm for finding a Hamiltonian cycle in a QT-graph; for an input graph on n vertices and m edges, our algorithm takes $O(\log n)$ time and requires $O(n+m)$ processors on the CREW PRAM model. In addition, we show that the problem of recognizing whether a QT-graph is a Hamiltonian graph and the problem of computing the Hamiltonian completion number of a nonHamiltonian QT-graph can also be solved in $O(\log n)$ time with $O(n+m)$ processors. Our algorithms rely on $O(\log n)$-time parallel algorithms, which we develop here, for constructing tree representations of a QT-graph; we show that a QT-graph G has a unique tree representation, that is, a tree structure which meets the structural properties of G. We also present parallel algorithms for other optimization problems on QT-graphs which run in $O(\log n)$ time using a linear number of processors.

Keywords: Parallel algorithms; Quasi-threshold graphs; Recognition; Tree representation; Hamiltonian cycles; Hamiltonian completion number; Complexity

1. Introduction

In this paper we consider finite undirected graphs with no loops nor multiple edges. Let G be such a graph with vertex set $V(G)$ and edge set $E(G)$. We say that G is a Hamiltonian graph if it has a spanning cycle (as opposed to the more usual definition which refers to spanning path); such a cycle is called a *Hamiltonian cycle* of G. The *Hamiltonian completion number* of the graph G is the minimum number of edges which need to be added to $E(G)$ to make G Hamiltonian [3,13]; we denote the Hamiltonian completion number of a graph G as $hcn(G)$. If G is a Hamiltonian graph, then $hcn(G)=0$.

A graph G is called *quasi-threshold*, or QT-graph for short, if G contains no induced subgraph isomorphic to P_4 or C_4 (cordless path or cycle on 4 vertices) [11,21,26,27]. The class of QT-graphs is a subclass of the class of cographs [8,9] and contains the class of threshold graphs [7,12,23].

Many researchers have devoted their work to the study of the class of QT-graphs. Wolk [26] called the members of this class *comparability graphs of trees* and gave structural and algorithmic characterizations. Golumbic [11] called these graphs *trivially perfect* with respect to a concept of "perfection". Ma et al. [21] established the name *quasi-threshold* graphs (QT-graphs) and studied algorithmic properties. A variety of sequential and parallel algorithms have been appeared in the literature for many interesting optimization and combinatorial problems on the class of QT-graphs and, also, the classes of cographs and threshold graphs. We present, here, efficient $O(\log n)$-time parallel algorithms for Hamiltonian problems on the class of QT-graphs.

1.1. Related research

The class of QT-graphs is a subclass of the well-known class of perfect graphs [5,12]; it is a very important class of graphs, since a number of problems, which are NP-complete in general, can be solved in polynomial time on its members. For the class of QT-graphs, Ma et al. [21] presented polynomial algorithms for a number of optimization problems. In particular,

E-mail address: stavros@cs.uoi.gr.

0743-7315/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2003.08.004

they described an $O(nm)$ time algorithm for the recognition problem, and polynomial-time algorithms for the Hamiltonian cycle problem and the bandwidth problems. They also gave a formula for the clique covering number and conditions for a QT-graph to be Hamiltonian. Yan et al. [29] stated important characterizations of these graphs and presented a linear-time algorithm for the recognition problem. They also proposed linear-time algorithms for the edge domination problem and the bandwidth problem.

Hamiltonian problems on QT-graphs have not been the focus of much research in parallel process environment. However, algorithmic solutions can be obtained from the variety of available sequential and parallel algorithms for these problems on cographs, a superclass of the class of QT-graphs. Cographs themselves were introduced in the early 1970s by Lerchs [18] who studied their structural and algorithmic properties. Lerchs has shown, among other properties, the following two very nice algorithmic properties: (i) cographs are exactly the P_4 restricted graphs, and (ii) cographs admit a unique tree representation, up to isomorphism, called *cotree*.

There are several algorithms for recognizing cographs and constructing their cotrees. A linear time sequential algorithm for recognizing a cograph, through the construction of its cotree, is given in [8]. It is known that cographs can also be efficiently (but not optimally) recognized in parallel by a polynomial number of processors in polylogarithmic time; see [10,16,19]. In fact, a nearly optimal parallel algorithm for the cograph recognition and cotree construction problem was developed by He [16] which, on a graph on n vertices and m edges, takes $O(\log^2 n)$ time and requires $O(n+m)$ processors on the CRCW PRAM model. Dahlhaus [10] also proposed a nearly optimal parallel recognition algorithm working in $O(\log^2 n)$ time with $O(n+m)$ processors on the CREW PRAM model. For the same problem, Lin and Olariu [19] presented a parallel algorithm working in $O(\log n)$ time with $O((n^2+nm)/\log n)$ processors on the EREW PRAM model. All the algorithms determine if a graph is a cograph, and if so construct its cotree.

Many interesting optimization problems in graph theory, which are NP-complete in general graphs, have polynomial sequential solutions and admit efficient or even optimal parallel algorithms in the class of cographs, and, thus, in the class of QT-graphs. Such problems, with a large spectrum of practical applications, include the coloring problem, the Hamiltonian cycle and Hamiltonian path problems, the minimum path cover problem, and many other ones. Lin et al. [20] presented an optimal sequential algorithm for determining the minimum path cover for a cograph, which exhibits a Hamiltonian cycle or path as well, if such a structure exists. Bodlaender and Möhring [4] proved that the pathwidth of a cograph equals its treewidth and proposed a linear-time algorithm to determine the pathwidth of a cograph. In a parallel environment, given the cotree of a cograph as input, both the Hamiltonian path and Hamiltonian cycle problems are solved in $O(\log^2 n)$ time with $O(n^2)$ processors [1], and the minimum path cover problem and the maximum matching problem are solved in $O(\log n)$ time with $O(n/\log n)$ processors [19a,22].

The cotree of a cograph is constructed in $O(\log^2 n)$ time with $O(n+m)$ processors [10,16], or in $O(\log n)$ time with $O((n^2+nm)/\log n)$ processors [19]; thus, the cotree construction dominates the time and/or processor complexity of the parallel algorithms in [1,20,22] for solving various Hamiltonian and optimization problems on cographs, and, thus, on QT-graphs. It follows that these parallel algorithms need, in total, either $O(\log^2 n)$ time or $O((n^2+nm)/\log n)$ processors, since they require the cotree as input, and not the standard adjacency-list representation of the input cograph.

1.2. Our results

In this paper we study the class of QT-graphs in more detail and show structural and algorithmic properties of its members. We prove that a QT-graph G has a unique tree representation, that is, a tree structure that meets the structural properties of G; we refer to this tree as *cent-tree* of the graph G and denote $T_c(G)$. We define a depth-first search traversal of the cent-tree $T_c(G)$, which we call h-dfs, and prove necessary and sufficient conditions for a QT-graph to be Hamiltonian. Consequently, by taking advantage of these properties and conditions, we construct efficient parallel algorithms for Hamiltonian problems on QT-graphs.

In particular, we first describe a parallel algorithm for the construction of the cent-tree of a QT-graph, which runs in $O(\log n)$ time using $O(n+m)$ processors on the CREW PRAM model. Then, we construct an algorithm for finding a Hamiltonian cycle of a QT-graph; our algorithm takes $O(\log n)$ time and requires $O(n+m)$ processors on the CREW PRAM model. In addition, we show that the problem of recognizing whether a QT-graph is a Hamiltonian graph and computing the Hamiltonian completion number of a nonHamiltonian QT-graph can also be solved in $O(\log n)$ time with $O(n+m)$ processors. We also present parallel algorithms for other optimization problems on QT-graphs which run in $O(\log n)$ time using a linear number of processors.

Our algorithms run on the CREW PRAM model of computation [2,17,25], and use a linear number of processors on QT-graphs with n vertices and m edges. More precisely, we present the following results:

(i) The cent-tree of a QT-graph can be constructed in $O(\log n)$ time with $O(n+m)$ processors.

(ii) A Hamiltonian cycle and a Hamiltonian completion edge set of a QT-graph can be constructed in $O(\log n)$ time with $O(n+m)$ processors.
(iii) Hamiltonian QT-graphs can be recognized in $O(\log n)$ time with $O(n+m)$ processors.
(iv) The Hamiltonian completion number of a non-Hamiltonian QT-graph can be computed in $O(\log n)$ time with $O(n+m)$ processors.
(v) Other optimization problems on QT-graphs can be solved in $O(\log n)$ time with $O(n+m)$ processors: the maximum clique problem, the maximum independent set problem, the clique cover problem and the coloring problem.

It is worth noting that the algorithms we present here are based on new algorithmic and structural properties of QT-graphs and are not parallel versions of existing sequential ones. Moreover, the input to all our algorithms is the given graph in its adjacency-list representation.

1.3. Organization of the paper

The paper is organized as follows. In Section 2 we characterize the class of QT-graphs in detail and show structural and algorithmic properties on the class of QT-graphs. In Section 3 we prove necessary and sufficient conditions for a QT-graph to be Hamiltonian. In Sections 4 we present a parallel algorithm for constructing the cent-tree representation of a QT-graph. Based on this representation and the conditions of Section 3, we present the main results of the paper in Sections 5 and 6; we describe parallel algorithms for finding a Hamiltonian cycle in a QT-graph, recognizing a Hamiltonian QT-graph, and computing the Hamiltonian completion number of a nonHamiltonian QT-graph. In Section 7 we show that other optimization problems on QT-graph can be efficiently solved in parallel. Finally, in Section 8 we conclude with a summary of our results and extensions.

2. Quasi-threshold graphs and their structures

Let G be such a graph with vertex set $V(G)$ and edge set $E(G)$. The neighborhood $N(x)$ of a vertex $x\in V(G)$ is the set of all the vertices of G which are adjacent to x. The closed neighborhood of x is defined as $N[x] \coloneqq \{x\}\cup N(x)$ [14]. Given a graph G, an edge $(x,y)=(y,x)$ of G can be classified as follows according to the relationship of closed neighborhoods: (x,y) is *free* if $N[x]=N[y]$; (x,y) is *semi-free* if $N[x]\subset N[y]$ (or $N[y]\subset N[x]$); and (x,y) is *actual* otherwise [15,24,23]. Obviously, the edge set $E(G)$ of a graph G can be partitioned into the three subsets of free edges, semi-free edges and of actual edges, respectively.

The subgraph of a graph G induced by a subset S of the vertex set $V(G)$ is denoted by $G[S]$. For a vertex subset S of G, we define $G-S \coloneqq G[V(G)-S]$.

The following lemma follows immediately from the fact that for every subset $S\subset V(G)$ and for a vertex $x\in S$, we have $N_{G[S]}[x]=N[x]\cap S$ and that $G-S$ is an induced subgraph.

Lemma 2.1 (Nikolopoulos [24]). *If G is a QT-graph, then for every subset $S\subset V(G)$, both $G[S]$ and $G[V(G)-S]$ are also QT-graphs.*

The following theorem provides important properties for the class of QT-graphs. For convenience, we define $cent(G)=\{x\in V(G)\mid N[x]=V(G)\}$.

Theorem 2.1 (Nikolopoulos [24]). *Let G be an undirected graph.*

(i) *G is a QT-graph if and only if every connected induced subgraph $G[S], S\subseteq V(G)$, satisfies $cent(G[S])\neq\emptyset$.*
(ii) *G is a QT-graph if and only if $G[V(G)-cent(G)]$ is a QT-graph.*
(iii) *Let G be a connected QT-graph. If $V(G)-cent(G[S])\neq\emptyset$, then $G[V(G)-cent(G)]$ contains at least two connected components.*

Let G be a connected QT-graph. Then $V_1 \coloneqq cent(G)$ is not an empty set by Theorem 2.1. Put $G_1 \coloneqq G$, and $G[V(G)-V_1]=G_2\cup G_3\cup\cdots\cup G_r$, where each G_i is a connected component of $G[V(G)-V_1]$ and $r\geqslant 3$. Then since each G_i is an induced subgraph of G, G_i is also a QT-graph, and so let $V_i \coloneqq cent(G_i)\neq\emptyset$ for $2\leqslant i\leqslant r$. Since each connected component of $G_i[V(G_i)-cent(G_i)]$ is also a QT-graph, we can continue this procedure until we get an empty graph. Then we finally obtain the following partition of $V(G)$:

$$V(G)=V_1+V_2+\cdots+V_k \quad \text{where } V_i=cent(G_i).$$

Moreover we can define a partial order $\preccurlyeq$ on the set $\{V_1,V_2,\ldots,V_k\}$ as follows:

$$V_i\preccurlyeq V_j \quad \text{if } V_i=cent(G_i) \text{ and } V_j\subseteq V(G_i).$$

It is easy to see that the above partition of the vertex set $V(G)$ of the QT-graph G possesses the following properties.

Theorem 2.2 (Nikolopoulos [24]). *Let G be a connected QT-graph, and let $V(G)=V_1+V_2+\cdots+V_k$ be the partition defined above; in particular, $V_1 \coloneqq cent(G)$. Then this partition and the partially ordered set $(\{V_i\},\preccurlyeq)$ have the following properties*:

(P1) *If $V_i\preccurlyeq V_j$, then every vertex of V_i and every vertex of V_j are joined by an edge of G.*

(P2) *For every* V_j, $cent(G[\{\bigcup V_i \mid V_i \preccurlyeq V_j\}]) = V_j$.

(P3) *For every two* V_s *and* V_t *such that* $V_s \preccurlyeq V_t$, $G[\{\bigcup V_i \mid V_s \preccurlyeq V_i \preccurlyeq V_t\}]$ *is a complete graph. Moreover, for every maximal element* V_t *of* $(\{V_i\}, \preccurlyeq)$, $G[\{\bigcup V_i \mid V_1 \preccurlyeq V_i \preccurlyeq V_t\}]$ *is a maximal complete subgraph of* G.

(P4) *Every edge with both endpoints in* V_i *is a free edge, and every edge with one endpoint in* V_i *and the other endpoint in* V_j, *where* $V_i \neq V_j$, *is a semi-free edge.*

The results of Theorem 2.2 provide structural properties for the class of QT-graphs. We shall refer to the structure that meets the properties of Theorem 2.2 as *cent-tree* of the graph G and denote it by $T_c(G)$. The cent-tree $T_c(G)$ of a QT-graph is a rooted tree; it has nodes $V_1, V_2, \ldots, V_k$, root $V_1 := cent(G)$, and every node V_i is either a leaf or has at least two children. Moreover, $V_s \preccurlyeq V_t$ if and only if V_s is an ancestor of V_t in $T_c(G)$. Thus, we can state the following result (Fig. 1).

Corollary 2.1. *A graph* G *is a* QT*-graph if and only if* G *has a cent-tree* $T_c(G)$.

Observation 2.1. Let G be a QT-graph and let $V = V_1 + V_2 + \cdots + V_k$ be the above partition of $V(G)$; $V_1 := cent(G)$. Let $S = \{v_s, v_{s+1}, \ldots, v_t, \ldots, v_q\}$ be a stable set such that $v_t \in V_t$ and V_t is a maximal element of $(V_i, \preccurlyeq)$ or, equivalently, V_t is a leaf node of $T_c(G)$, $s \leqslant t \leqslant q$. It is easy to see that S has the maximum cardinality $\alpha(G)$ among all the stable sets of G. On the other hand, the sets $\{\bigcup V_i \mid V_1 \preccurlyeq V_i \preccurlyeq V_t\}$, for every maximal element V_t of $(V_i, \preccurlyeq)$, provide a clique cover of size $\kappa(G)$ which is the smallest possible clique cover of G; that is $\alpha(G) = \kappa(G)$. Based on Theorem 2.2 or, equivalently, on the properties of the cent-tree of G, it is easy to show that the clique number $\omega(G)$ equals the chromatic number $\chi(G)$ of the graph G; that is, $\chi(G) = \omega(G)$.

Let V_i and V_j be disjoint vertex sets of the above partition of $V(G)$ of a QT-graph G. We say that V_i and V_j are *adjacent* if either $V_i \preccurlyeq V_j$ or $V_j \preccurlyeq V_i$; otherwise, we say that V_i and V_j are *non-adjacent*. Throughout the paper, we call *clique-adjacent* (resp. *independent*), and denote $V_i \approx V_j$ (resp. $V_i \asymp V_j$), two adjacent (resp. non-adjacent) vertex sets V_i and V_j of the partition of $V(G)$.

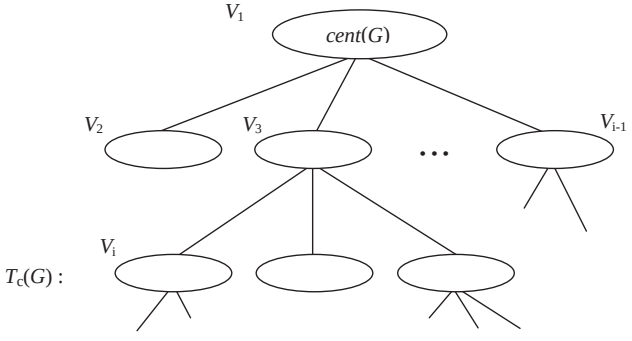

Fig. 1. The typical structure of the cent-tree $T_c(G)$ of a QT-graph.

3. Hamiltonian QT-graphs

Let G be a QT-graph G and let $V_1, V_2, \ldots, V_k$ be the nodes of the cent-tree $T_c(G)$ rooted at $r_c = V_1$. We consider a node V_i of $T_c(G)$ and let $V_{i1}, V_{i2}, \ldots, V_{ip_i}$ be its children, $1 \leqslant i \leqslant k$; note that $p_i \geqslant 2$ if V_i is not a leaf; otherwise $p_i = 0$. We assign a label H-label (V_i) to the node V_i, $1 \leqslant i \leqslant k$, which we compute as follows:

$$H\text{-}label(V_i) = \begin{cases} |V_i| - p_i & \text{if } V_i \text{ is the root of the tree,} \\ |V_i| - p_i + 1 & \text{if } V_i \text{ is an internal node, and,} \\ 0 & \text{if } V_i \text{ is a leaf,} \end{cases}$$

where p_i is the number of children of the node V_i. Fig. 2 depicts a node V_i of a cent-tree along with its four children V_{i1}, V_{i2}, V_{i3} and V_{i4}; here, we have $H\text{-label}(V_i) = 1$ if V_i is the root of the tree or $H\text{-label}(V_i) = 2$ if V_i is an internal node, and $H\text{-label}(V_{i1}) = 1$, $H\text{-label}(V_{i2}) = -1$, $H\text{-label}(V_{i3}) = 0$ and $H\text{-label}(V_{i4}) = 0$. We shall show that G is a Hamiltonian QT-graph if $H\text{-label}(V_i) \geqslant 0$ for each node $V_i \in T_c(G)$.

Let V_i be an internal node of the cent-tree $T_c(G)$ such that $H\text{-label}(V_i) \geqslant 0$ and let $V_{i1}, V_{i2}, \ldots, V_{ip_i}$ be its children, $p_i \geqslant 2$. Since V_i is an internal node and $H\text{-label}(V_i) \geqslant 0$, it has more than $p_i - 1$ vertices; let $list(V_i) = (v_{i1}, \ldots, v_{i(p_i-1)}, v_{ip_i}, \ldots, v_{in_i})$ be the list of the vertices of the node V_i, where $n_i \geqslant p_i - 1$.

We define the list $a\text{-vertices}(V_i) := (v_{ip_i}, v_{i(p_i+1)}, \ldots, v_{in_i})$; the elements of the list $a\text{-vertices}(V_i)$ are called *available vertices* of the node V_i. If V_i is the root of the cent-tree then $a\text{-vertices}(V_i) := (v_{i(p_i+1)}, v_{i(p_i+2)}, \ldots, v_{in_i})$. In Fig. 2, for the internal node V_i we have $a\text{-vertices}(V_i) = \{u, v\}$.

Let $V_{f(1)}, V_{f(2)}, \ldots, V_{f(t)}$ be the left-to-right order listing of the leaves of the cent-tree $T_c(G)$, and let $V_{a(i)}$ be the lowest common ancestor of the nodes $V_{f(i)}$ and $V_{f(i+1)}$, where $1 \leqslant i \leqslant t-1$. We define a sequence of

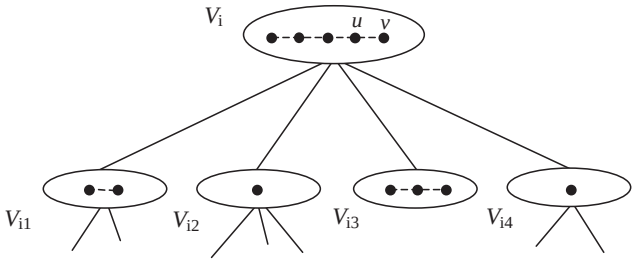

Fig. 2. A node of the cent-tree $T_c(G)$ of a QT-graph along with its four children; the vertices of each node of $T_c(G)$ are denoted by black disks.

nodes of the cent-tree $T_c(G)$, which we call *h-sequence* and denote h-sequence(T_c), as follows:

$$h\text{-}sequence(T_c) = (V_{f(1)}, V_{a(1)}, V_{f(2)}, V_{a(2)}, \ldots, V_{f(t)}, V_{a(t)} = V_1),$$

where V_1 is the root of the tree $T_c(G)$ and t is the number of leaves in $T_c(G)$; the length of the h-sequence of $T_c(G)$ is $2t$.

By definition there exists no pair $\{V_{f(i)}, V_{f(j)}\}$ of elements of the h-sequence(T_c) such that $V_{f(i)} = V_{f(j)}$ for $i \neq j$ and $1 \leqslant i, j \leqslant t$. On the other hand, there may exist elements $V_{a(i_1)}, V_{a(i_2)}, \ldots, V_{a(i_{p-1})}$ such that $V_{a(i_1)} = V_{a(i_2)} = \cdots = V_{a(i_{p-1})} = V_i$, where V_i is an internal node of $T_c(G)$ and p is equal to the number of children of V_i. Let $a(i_1)$ and $a(i_{p-1})$ be the indices of the leftmost and rightmost occurrence of V_i in h-sequence(T_c), and let $a(i_1) < a(i_2) < \cdots < a(i_{p-1})$. We say that $V_{a(i_1)}$ is the first occurrence of V_i, $V_{a(i_2)}$ is the second occurrence of V_i, and so on; $V_{a(i_{p-1})}$ is the last (rightmost) occurrence of V_i in h-sequence(T_c). Based on the structure of the cent-tree $T_c(G)$ and the fact that each internal node of $T_c(G)$ has at least two children we can easily conclude that each internal node of $T_c(G)$ appears at least once in the h-sequence. Thus, we can prove the following result.

Proposition 3.1. *All the nodes of the cent-tree $T_c(G)$ of a QT-graph G are appeared in the sequence h-sequence(T_c). Moreover, two consecutive nodes in h-sequence(T_c) are clique-adjacent.*

Proof. Let h-sequence$(T_c) = (V_{f(1)}, V_{a(1)}, V_{f(2)}, \ldots, V_{a(t)} = V_1)$ and let $V_1, V_2, \ldots, V_k$ be the nodes of the cent-tree $T_c(G)$. By definition, the nodes $V_{f(1)}, V_{f(2)}, \ldots, V_{f(t)}$ form a left-to-right order listing of the leaves of the cent-tree $T_c(G)$ and the node $V_{a(i)}$ is the lowest common ancestor of the nodes $V_{f(i)}$ and $V_{f(i+1)}$, where $1 \leqslant i \leqslant t-1$. Thus, all the nodes of the cent-tree $T_c(G)$ appeared in h-sequence(T_c).

Two consecutive nodes in h-sequence(T_c), say, $V_{f(i)}$, $V_{a(i)}$ or $V_{a(i)}$, $V_{f(i+1)}$, are clique-adjacent since the node $V_{a(i)}$ is an ancestor of both nodes $V_{f(i)}$ and $V_{f(i+1)}$ in $T_c(G)$, that is, $V_{a(i)} \preccurlyeq V_{f(i)}$ and $V_{a(i)} \preccurlyeq V_{f(i+1)}$; see Theorem 2.2 (property P3). □

Let $V_1, V_2, \ldots, V_k$ be the nodes of the cent-tree $T_c(G)$ of a QT-graph G and let h-sequence(T_c) be the h-sequence of $T_c(G)$. We next use a depth-first search (dfs) traversal strategy for visiting the vertices of the graph G (i.e., the vertices in each node V_i, $1 \leqslant i \leqslant k$) and building a depth-first tree. We shall use the h-sequence for the process of selecting the next unvisited vertex; note that in the standard dfs traversal when we have a choice of vertices to visit, we select them in alphabetical order. Based on the h-sequence for the selection process, we describe a dfs traversal, which, hereafter, we shall call *h-dfs* traversal; it works as follows:

The h-dfs traversal of the cent-tree $T_c(G)$:

(i) Compute the h-sequence $(V_{f(1)}, V_{a(1)}, V_{f(2)}, \ldots, V_{f(t)}, V_{a(t)} = V_1)$ of the cent-tree $T_c(G)$;
(ii) Select an arbitrary vertex v from $V_{f(1)}$ as starting vertex; visit v and mark it "visited" (initially, all vertices of G are marked "unvisited");
(iii) Visit in turn each unvisited vertex v_i of $V_{f(i)}$, $1 \leqslant i \leqslant t$, and mark v_i "visited";
(iv) Once all the vertices of $V_{f(i)}$ have been visited, go to node $V_{a(i)}$ and do the following:
- if there exists an unvisited vertex u_i in $V_{a(i)} = V_i$, then select u_i, mark u_i "visited" and if $V_{a(i)} = V_i$ is its rightmost occurrence in h-sequence(T_c), then select in turn each unvisited vertex of $V_{a(i)}$ and mark it "visited";
- if there exists no unvisited vertex u_i in $V_{a(i)} = V_i$, then do nothing;

(v) Continue until the last vertex u of the rood node $V_{a(t)} = V_1$ becomes a visited vertex; that is, if i is less that t, then increase the i by 1 and execute Steps (iii) and (iv).

In Fig. 3 we show the h-dfs traversal of a cent-tree $T_c(G)$ on six nodes; its h-sequence is the following: $(V_{f(1)}, V_{a(1)}, V_{f(2)}, V_{a(2)}, V_{f(3)}, V_{a(3)}, V_{f(4)}, V_{a(4)})$, where $V_{a(1)} = V_{a(2)} = V_{a(3)} = V_{a(4)} = V_1$. The vertices v and u are the first and the last vertices of the lists list$(V_{f(1)})$ and list(V_1), respectively.

We consider connected undirected QT-graphs and, thus, the h-dfs forest of such a graph G contains only one tree. Moreover, it is obvious that if each vertex of the h-dfs tree of G rooted at $v \in V(G)$ has at most one child, then G contains a Hamiltonian path beginning with vertex v (it is the path from the root v to the unique leaf u); G contains a Hamiltonian cycle if the root v of the h-dfs tree and its unique leaf u are adjacent in G. We next prove the following result.

Lemma 3.1. *Let G be a QT-graph and let $V_1, V_2, \ldots, V_k$ be the nodes of its cent-tree $T_c(G)$. The QT-graph G is a Hamiltonian graph if H-label$(V_i) \geqslant 0$ for each node $V_i \in T_c(G)$.*

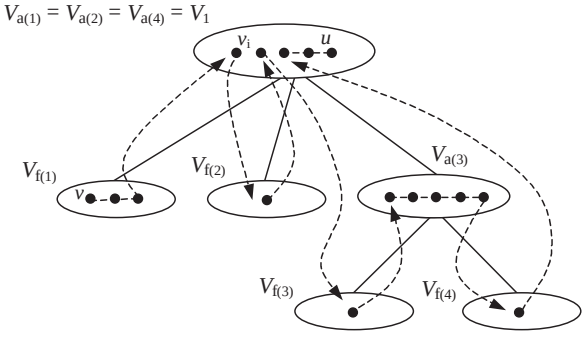

Fig. 3. The h-dfs traversal of a cent-tree $T_c(G)$ of a QT-graph G.

Proof. Let $V_1, V_2, \ldots, V_k$ be the nodes of the cent-tree $T_c(G)$ of the QT-graph G rooted at V_1, and let $(V_{f(1)}, V_{a(1)}, V_{f(2)}, \ldots, V_{f(t)}, V_{a(t)} = V_1)$ be the h-sequence of $T_c(G)$. By definition, the internal node V_i appears $p_i - 1$ times in the sequence h-sequence(T_c), where p_i is the number of children V_i in the cent-tree $T_c(G)$; let $V_{a(i_1)} = V_{a(i_2)} = \cdots = V_{a(i_{p-1})} = V_i$, $1 \leqslant i \leqslant k$.

Since H-label$(V_i) \geqslant 0$, it follows that the node V_i contains at least $p_i - 1$ vertices; it contains p_i vertices if V_i is the root V_1 of the cent-tree $T_c(G)$. Let n_i be the number of vertices in $V_i \in T_c(G)$ and let list$(V_i) = (v_{i1}, \ldots, v_{i(p_i-1)}, v_{ip_i}, \ldots, v_{in_i})$.

We select a vertex v from $V_{f(1)}$ and we perform an h-dfs traversal starting at vertex v. Since each node V_i contains at least $p_i - 1$ vertices (p_i vertices if $V_i = V_1$), it follows that after visiting the vertices of the node $V_{f(i)}$ there exists at least one unvisited vertex in $V_{a(i)}$ and, thus, the h-dfs always selects the next vertex from $V_{a(i)}$, $1 \leqslant i \leqslant t-1$; this is also true for the nodes $V_{f(t)}$ and $V_{a(t)} = V_1$. On the other hand, the nodes $V_{a(i)}$ and $V_{f(i+1)}$ are clique-adjacent; see Proposition 3.1. Thus, the h-dfs tree has the property that each node has at most one child; that is, G contains a Hamiltonian path. Since V_1 is the root of $T_c(G)$, we have that $V_1 \preccurlyeq V_{f(1)}$, that is, $V_{f(1)}$ and V_1 are clique-adjacent. Thus, G contains a Hamiltonian cycle. □

We consider now the case where the cent-tree $T_c(G)$ of a Hamiltonian QT-graph has nodes, say, V_i and V_j, such that $V_i \leqslant V_j$ and H-label$(V_i) > 0$ and H-label$(V_j) < 0$. Let u be an available vertex of the node V_i. Hereafter, when we say that we v-move the available vertex u from the node V_i to node V_j, we mean that (i) we delete the vertex u from node V_i and (ii) we add it to node V_j.

From the structure of the cent-tree $T_c(G)$ of a QT-graph, it is easy to see that if we apply a v-move operation to nodes V_i and V_j, then the resulting tree has the Property (P3): for every two nodes V_s and V_t such that $V_s \preccurlyeq V_t$, $G[\{\bigcup V_i \mid V_s \preccurlyeq V_i \preccurlyeq V_t\}]$ is a complete graph. Obviously, if V_t is a maximal element of $(\{V_i\}, \preccurlyeq)$, then after applying a v-move operation the graph $G[\{\bigcup V_i \mid V_1 \preccurlyeq V_i \preccurlyeq V_t\}]$ may not be a maximal complete subgraph of G.

Consider the tree that results from the cent-tree $T_c(G)$ of a QT-graph after applying some v-move operations on appropriate nodes so that each node V_i of that tree has H-label greater than or equal to 0; we call such a tree *h-tree* and denote it by $T_h(G)$. Then, we prove the following result.

Theorem 3.1. *Let G be a QT-graph and let $V_1, V_2, \ldots, V_k$ be the nodes of its cent-tree $T_c(G)$. The QT-graph G is a Hamiltonian graph if and only if either H-label$(V_i) \geqslant 0$ for each node $V_i \in T_c(G)$ or we can construct an h-tree $T_h(G)$.*

Proof. The if implication follows directly from Lemma 3.1 since either H-label$(V_i) \geqslant 0$ for each node V_i of the cent-tree $T_c(G)$ or H-label$(V_i) \geqslant 0$ for each node V_i of the h-tree $T_h(G)$, $1 \leqslant i \leqslant k$.

Suppose now that there exists a node in $T_c(G)$, say, V_i, such that H-label$(V_i) < 0$, and also suppose that there exists no ancestor of the node V_i in $T_c(G)$, say, V_j, with available vertices, that is, H-label$(V_j) \leqslant 0$. Thus, we cannot construct an h-tree $T_h(G)$.

Let $V_{i1}, V_{i2}, \ldots, V_{ip_i}$ be the children of the node V_i in the cent-tree $T_c(G)$, and let n_i be the number of vertices of V_i, that is, list$(V_i) = (v_{i1}, v_{i2}, \ldots, v_{in_i})$. Since H-label$(V_i) < 0$, exactly one of the following cases holds:

Case (i): $n_i < p_i$, if V_i is the root of the cent-tree $T_c(G)$;

Case (ii): $n_i < p_i - 1$, if V_i is an internal node of the cent-tree $T_c(G)$;

Let $(V_{f(1)}, V_{a(1)}, \ldots, V_{a(t)} = V_1)$ be the h-sequence of $T_c(G)$. We select a vertex v from $V_{f(1)}$ and we perform an h-dfs traversal to the vertices of G starting at vertex v. We consider the two cases:

Case (i): $V_i = V_1$ is the root of the cent-tree $T_c(G)$, and $n_1 < p_1$. From the structure of the cent-tree $T_c(G)$, we have that the subgraph of G induced by the vertices of the subtrees of $T_c(G)$ rooted at $V_{11}, V_{12}, \ldots, V_{1p_1}$, that is, the graph $G - V_1$, contains p_1 connected components, say, $C_{11}, C_{12}, \ldots, C_{1p_1}$. We consider the best case where each induced graph $G(C_{1i})$, $1 \leqslant i \leqslant p_1$, contains a Hamiltonian path (or cycle). Since $n_1 < p_1$, it follows that the p_1 paths (path cover of $G - V_1$) cannot be extended to a cycle using the n_1 vertices of the node V_1; the p_1 paths can be extended to a (Hamiltonian) path if $n_1 = p_1 - 1$. Thus, the whole graph G does not contain a Hamiltonian cycle. Note that, the vertex $v_{1n_1} \in V_1$ has at least two children in the h-dfs tree if $n_1 < p_1 - 1$, while it has exactly one child u, if $n_1 = p_1 - 1$; the vertex u belongs to a node of the subtree of $T_c(G)$ rooted at V_{1p_1}.

Case (ii): V_i is an internal node, and $n_i < p_i - 1$. Again, from the structure of the cent-tree $T_c(G)$, we have that the subgraph of G induced by the vertices of the subtrees of $T_c(G)$ rooted at $V_{i1}, V_{i2}, \ldots, V_{ip_i}$, contains p_i connected components, say, $C_{i1}, C_{i2}, \ldots, C_{ip_i}$. From case (i), we have that the subgraph of G induced by the vertices of the subtree of $T_c(G)$ rooted at V_i does not contain a Hamiltonian cycle since $n_i < p_i - 1$; it does not also contain a (Hamiltonian) path. Since there exists no ancestor of the node V_i in $T_c(G)$ with available vertices, it follows that the paths cannot be extended to a (Hamiltonian) path. Thus, even the node V_1 contains $n_1 = p_1$ vertices, the whole graph G does not contain a Hamiltonian cycle. Note that, the vertex $v_{in_i} \in V_i$ has at least two children in the h-dfs tree.

Thus, in both cases (i) and (ii) the QT-graph G does not contain a Hamiltonian cycle, and the theorem is proved. □

4. Construction of the cent-tree of a QT-graph

The characterizations provided by Theorem 2.2 enable us to describe a parallel algorithm for constructing the cent-tree of a QT-graph.

4.1. The degree-tree

Let G be a QT-graph and let $T_c(G)$ be its cent-tree with node set $\{V_1, V_2, \ldots, V_k\}$ and root V_1. We have proved that if the node V_i is an ancestor of the node V_j in the cent-tree $T_c(G)$, then V_i and V_j are clique-adjacent; see Theorem 2.2. Thus, if the sequence of nodes $(V_1, V_2, \ldots, V_i)$ forms a path from the root V_1 of the cent-tree to a node V_i, then $d(V_1) > d(V_2) > \cdots > d(V_i)$, where $d(V_i)$ denotes the degree of the vertices of G that belong to node V_i; recall that all the vertices of G that belong to node V_i have the same degree and each internal node of the cent-tree of G has at least two children. It follows that if $\{v_1, v_2, \ldots, v_p\}$ is a clique in a QT-graph, then $d(v_1) \geqslant d(v_2) \geqslant \cdots \geqslant d(v_p)$, $1 \leqslant p \leqslant n$.

Based on this property, we describe an algorithm that produces a tree representation of a QT-graph; see also [29]. We call this tree *degree-tree* of the QT-graph G and we denote it by $T_d(G)$. The details of the algorithm are given as follows:

(1) Sort the vertices $v_1, v_2, \ldots, v_n$ of G according to their degrees; let $D = (v_1, v_2, \ldots, v_n)$ be a sequence such that $d(v_1) \geqslant d(v_2) \geqslant \cdots \geqslant d(v_n)$.
(2) Construct the tree $T_d(G)$ in the following manner:
   (i) set $V(T_d) = \{v_1, v_2, \ldots, v_n\}$ and $E(T_d) = \emptyset$;
   (ii) for every vertex $v_i \in D$, $2 \leqslant i \leqslant n$, find the vertex v_k, if it exists, such that k is the maximum index satisfying $1 \leqslant k < i$ and (v_k, v_i) is an edge in G; add the edge (v_k, v_i) in the edge set $E(T_d)$.
(3) Root the tree $T_d(G)$ at vertex $r = v_1$; the resulting tree is the degree-tree $T_d(G)$.

We call D-TREE-CON the above construction algorithm, and show that it can be easily implemented to run on a PRAM model of computation.

Time and processor complexity. The algorithm takes as input a QT-graph G on n vertices and m edges, where G is given in an adjacency-list representation, and constructs the degree-tree $T_d(G)$; its time and processor complexity is analyzed as follows:

Step 1: It is known that the degree of each vertex of a graph G on n vertices and m edges can be computed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model; G is given in an adjacency-list representation. It is also known that n elements can be sorted in $O(\log n)$ time with $O(n)$ processors on the same model of computation [2,17,25].

Step 2: Let $D = (v_1, v_2, \ldots, v_n)$ be the vertex sequence computed in Step 1, and let $N(v_i) = \{u_1, u_2, \ldots, u_{d_i}\}$ be the set of vertices adjacent to v_i, $1 \leqslant i \leqslant n$. For each vertex v_i we compute the vertex u, if it exists, such that: $u \in N(v_i)$ and u is the nearest vertex to the left of v_i in the sequence D. This computation can be carried out through the general prefix computation (GPC, see [2]) in $O(\log n)$ time with $O(n)$ processors on the CREW PRAM model.

Step 3: The problem of rooting the tree $T_d(G)$ at the vertex $r = v_1$ can be solved in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM using the well-known Euler-tour technique [2,17,25]. Thus, this step can be performed within the stated bounds.

Taking into consideration the time and processor complexity of each step of the construction algorithm D-TREE-CON, we can state the following result.

Lemma 4.1. *The degree-tree of a QT-graph can be constructed in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model.*

4.2. The cent-tree construction algorithm

Based on the structural properties of the degree-tree $T_d(G)$ of a QT-graph G, we next present a parallel algorithm for the construction of the cent-tree $T_c(G)$ of the graph G.

We observe that, a vertex u and its parent $p(u)$ belong to the same node $V_i \in T_c(G)$ if and only if u is a unique child of the vertex $p(u)$ in the degree-tree $T_d(G)$; see Fig. 4. Let $u_2, \ldots, u_k$ be the vertices of the degree-tree $T_d(G)$ with the property that their parents have at least two children, and let $R = \{r = u_1, u_2, \ldots, u_k\}$, where r is the

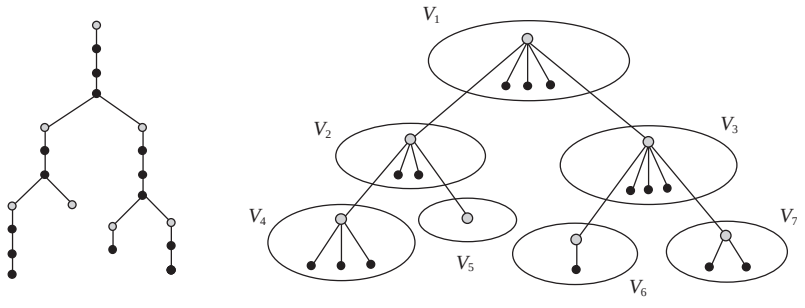

Fig. 4. The degree-tree $T_d(G)$ and the cent-tree $T_c(G)$ of a QT-graph G.

root of the tree $T_d(G)$; the vertices of R of the tree $T_d(G)$ of Fig. 4 are denoted by gray disks.

Let $V_1, V_2, \ldots, V_k$ be the nodes of the cent-tree $T_c(G)$ rooted at V_1. It is easy to see that $u_i \in V_i$, $1 \leqslant i \leqslant k$. The node V_1 is the root of the cent-tree and the node $V_i = \{u_i\}$ has parent the node $V_j = \{u_j\}$ in $T_c(G)$ if u_j is the least ancestor of u_i in $T_d(G)$ which belongs to R. The vertex $u \notin R$ of the graph G belongs to the node V_i if the vertex u_i is the least ancestor of u in $T_d(G)$, $1 \leqslant i \leqslant k$; see Fig. 4.

Based on the above observations and properties, we describe the following parallel algorithm; it takes as input a QT-graph G and produces the cent-tree of the QT-graph G.

Algorithm Cent-Tree-Construction (CENT-TREE-CON):

Input: a quasi-threshold graph G on n vertices and m edges.

Output: the cent-tree $T_c(G)$ of the input graph G.

1. Compute the degree-tree $T_d(G)$ and let $V(T_d) = \{r = v_1, v_2, \ldots, v_n\}$;
2. For each vertex $v_i \in V(T_d)$, $1 \leqslant i \leqslant n$, do in parallel
   if v_i is the root r of the tree or its parent $p(v_i)$ has more than one child,
   then set $color(v_i) \leftarrow$ red; otherwise $color(v_i) \leftarrow$ black;
   let $R = \{r = u_1, u_2, \ldots, u_k\}$ be the set of the red vertices of $T_d(G)$, $k \geqslant 1$;
3. For each vertex $v_i \in V(T_d) - R$, $2 \leqslant i \leqslant n$, do in parallel
   find the least ancestor u_i of vertex v_i with read color and set $p(v_i) \leftarrow u_i$;
   let $T_g(G)$ be the resulting tree rooted at $r = u_1$; we set $p(r) \leftarrow r$;
4. For each red vertex $u_i \in T_g(G)$ construct a node set V_i and set $V_i \leftarrow \{u_i\}$, $1 \leqslant i \leqslant k$;
5. Construct the tree graph $T_c(G)$, in parallel, as follows:
   5.1 Set $V(T_c) \leftarrow \{r_c = V_1, V_2, \ldots, V_k\}$;
   5.2 For each red vertex $u_i \in T_g(G)$, $2 \leqslant i \leqslant k$, do in parallel
      if u_j is the parent of u_i, then add the edge (V_i, V_j) into $E(T_c)$;
6. Compute the vertices of each node $V_1, V_2, \ldots, V_k$ of the tree $T_c(G)$ as follows:
   for each black vertex $v_i \in T_g(G)$, $2 \leqslant i \leqslant k$, do in parallel
      if u_j is the parent of v_i, then add the vertex v_i into node set V_j;
7. Root the tree $T_c(G)$ at node V_1; the rooted tree $T_c(G)$ is the cent-tree of the input graph G;

Time and processor complexity. We next compute the time and processor complexity of the proposed parallel algorithm for the construction of the cent-tree of a QT-graph. Its step-by-step analysis is as follows:

Step 1: The degree-tree $T_d(G)$ of a QT-graph on n vertices and m edges is constructed in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model; see Lemma 4.1.

Step 2: Let $v_1, v_2, \ldots, v_n$ be the vertices of the degree-tree $T_d(G)$ rooted at $r = v_1$. Obviously, the parent $p(v_i)$ of a vertex $v_i \in T_d(G)$, $2 \leqslant i \leqslant n$, has more than one child if there exist a vertex v_j such that $p(v_i) = p(v_j)$. Let $p(v_{i(1)}), p(v_{i(2)}), \ldots, p(v_{i(n)})$ be the sorted sequence of the parents of the vertices of $T_d(G)$; we set $p(v_{i(0)}) := 0$ and $p(v_{i(n+1)}) := n+1$. Then, the vertex $v_{i(j)}$ has more than one child if $p(v_{i(j-1)}) \neq p(v_{i(j)})$ and $p(v_{i(j)}) = p(v_{i(j+1)})$, $1 \leqslant j \leqslant n$. Since the sorting problem and the array packing problem on n elements can be solved in $O(\log n)$ time with $O(n)$ processors on the EREW PRAM model [2,17], this step can be executed within the same time and processor bounds.

Step 3: The tree $T_g(G)$ can be computed by using the pointer jumping technique on the degree-tree $T_d(G)$; for each vertex $v_i \in T_d(G)$, $1 \leqslant i \leqslant n$ such that $p(v_i)$ is a black vertex, do the following: set $p(v_i) := p(p(v_i))$; continue, until $p(v_i)$ is a red vertex, for every $v_i \in T_d(G)$ (this is the well-known parallel prefix algorithm [2,17]). Thus, the tree $T_g(G)$ can be constructed in $O(\log n)$ time with $O(n)$ processors on the CREW PRAM model.

Steps 4 *and* 5: It is easy to see that the node sets $V_1, V_2, \ldots, V_k$, and, thus, the set $V(T_c)$, can be computed in $O(1)$ time with $O(k)$ processors on the EREW PRAM model. The pair (V_i, V_j) is added into $E(T_c)$, if the vertex $u_i \in V_i$ has parent the vertex $u_j \in V_j$ in the tree $T_g(G)$. Thus, the computation of the edge set $E(T_c)$ requires $O(1)$ parallel time and $O(k)$ processors on the CREW PRAM model.

Step 6: It is easy to see that the elements of the node sets $V_1, V_2, \ldots, V_k$ can be computed from the tree $T_g(G)$; the vertex v belongs to V_i if the representative u_i of the set V_i is the parent of v in the tree $T_g(G)$. This computation can be carry out by using the sorted sequence of the parents of the vertices of the tree $T_d(G)$; it needs $O(\log n)$ time and $O(n)$ processors on the EREW PRAM model.

Step 7: We solve the problem of rooting $T_c(G)$ at the vertex V_1 by applying the Euler tour technique on the tree graph $T_c(G)$; it takes $O(\log n)$ time and requires $O(n/\log n)$ processors on the EREW PRAM model [2,17,25].

Therefore, from the previous step-by-step analysis, it follows that the construction algorithm CENT-TREE-CON runs in $O(\log n)$ time using a total of $O(n+m)$ processors on the CREW PRAM model of computation. Thus, we have proved the following result.

Theorem 4.1. *The cent-tree of a QT-graph can be constructed in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model.*

5. Finding a Hamiltonian cycle in a QT-graph

In this section, we first describe a parallel algorithm which constructs an h-tree $T_h(G)$ of a Hamiltonian QT-graph G. Recall that an h-tree of a QT-graph G has the property that H-label$(V_i) \geqslant 0$ for each node $V_i \in T_h(G)$. Next, we present a parallel algorithm which produces a Hamiltonian cycle of G; it works on the cent-tree $T_c(G)$ if H-label$(V_i) \geqslant 0$, for each node $V_i \in T_c(G)$; otherwise, it works on an h-tree $T_h(G)$ (see Theorem 3.1).

5.1. Construction of an H-tree

Consider the case where the cent-tree of a Hamiltonian QT-graph G has nodes, say, $V'_1, V'_2, \ldots, V'_{k'}$, with H-labels less than zero. In this case, we are interested in constructing an h-tree $T_h(G)$ from the cent-tree $T_c(G)$. An h-tree can be constructed by moving available vertices from certain nodes of $T_c(G)$ to nodes $V'_1, V'_2, \ldots, V'_{k'}$ (see Section 3). Recall that, if a Hamiltonian QT-graph G has a node, say, V_i, such that H-label$(V_i) < 0$, then there exists an ancestor V_j of V_i in the cent-tree $T_c(G)$ such that H-label$(V_j) > 0$; that is, the node V_j contains available vertices.

In more detail, an h-tree $T_h(G)$ can be constructed in the following manner: First, we determine the nodes of the cent-tree $T_c(G)$ with H-labels greater than zero and the available vertices which they contain. Next we determine the nodes $V'_1, V'_2, \ldots, V'_{k'}$ of the cent-tree $T_c(G)$, such that H-label$(V'_i) < 0$, and we v-move $|H\text{-label}(V'_i)|$ available vertices to node V'_i so that H-label$(V'_i) = 0$, where $1 \leqslant i \leqslant k'$. To this end, we add $d'_i = |H\text{-label}(V'_i)|$ dummy vertices, say, $v'_{i1}, v'_{i2}, \ldots, v'_{id'_i}$, to node V'_i, and we assign to each dummy vertex v'_{ij} an available vertex from an ancestor V_j of the node V'_i; (see Fig. 5; the available and the dummy vertices are denoted by grey and white disks, respectively). We call *m-vertex* the available vertex which has been assigned to a dummy vertex. Since G is a Hamiltonian QT-graph, we can always construct an h-tree $T_h(G)$ (see Theorem 3.1), and, thus, there is an one-to-one correspondence between the m-vertices and the dummy vertices. Finally, we v-move the m-vertices to appropriate nodes of the cent-tree and delete the dummy vertices from the cent-tree.

It is clear that the crucial factor of the above h-tree construction algorithm is the process of assignment available vertices to dummy vertices. We next show that this process can be efficiently implemented using the h-dfs traversal strategy and simple algorithmic techniques on certain nodes and vertex lists. This, in turn, implies an efficient implementation of the h-tree construction algorithm. We proceed as follows:

Step (A): *Construction of the available-dummy tree* (*ad-tree*)

(1) Determine, first, the nodes V_i of the cent-tree $T_c(G)$ such that H-label$(V_i) > 0$, compute the available vertices of V_i, and paint them with grey color; initially, all the vertices of $T_c(G)$ are black. Next, determine the nodes V'_i of the cent-tree $T_c(G)$ such that H-label$(V'_i) < 0$ and add a list of d'_i dummy vertices $(v'_{i1}, v'_{i2}, \ldots, v'_{id'_i})$ to the node V'_i, $1 \leqslant i \leqslant k'$; paint the dummy vertices of V'_i with white color. Let $T'_c(G)$ be the resulting tree.
(2) Delete from $T'_c(G)$ the nodes V_i such that H-label$(V_i) = 0$, that is, the node V_i contains only black vertices. Then, delete from the remaining nodes of $T'_c(G)$ all the black vertices. Now, the nodes of the tree $T'_c(G)$ contain either available or dummy vertices; a node with available (resp. dummy) vertices is called a-node (resp. d-node).
(3) For each node V_i of the tree $T'_c(G)$ (except from the root) do the following: if the parent $p(V_i)$ of the node V_i is a d-node, then determine the lowest a-node ancestor V_j of the node V_i in $T'_c(G)$, and set $p(V_i) := V_j$. Now, all the d-nodes of the resulting tree $T'_c(G)$ are leaves.
(4) For each a-node V_i of the tree $T'_c(G)$ compute two vertex lists, namely a-list(V_i) and d-list(V_i), as follows: a-list(V_i) contains the available vertices of V_i and d-list(V_i) contains the dummy vertices of all the d-node children of V_i.
(5) Delete from $T'_c(G)$ the d-nodes. The resulting tree is called *ad*-tree (available-dummy tree) and denoted by $T_{ad}(G)$.

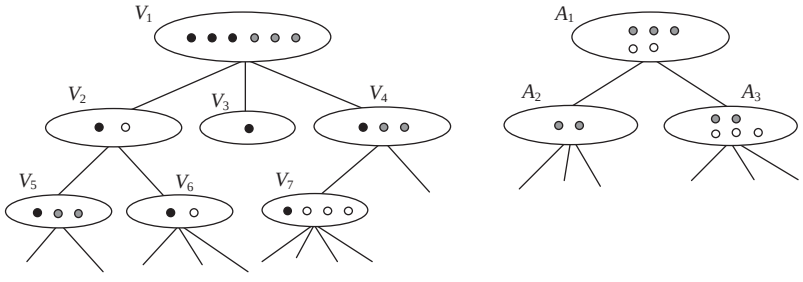

Fig. 5. The tree $T'_c(G)$ and the *ad*-tree $T_{ad}(G)$ of a QT-graph G.

Let $A_1, A_2, \ldots, A_t$ be the nodes of the *ad*-tree $T_{ad}(G)$ constructed by the above algorithm, and let $a\text{-list}(A_i) = (v_{i1}, v_{i2}, \ldots, v_{ia_i})$ be the list of the available vertices of the node A_i and $d\text{-list}(A_i) = (v'_{i1}, v'_{i2}, \ldots, v'_{id_i})$ be the list of the available vertices of A_i, $1 \leqslant i \leqslant t$.

Observation 5.1. From the construction of the *ad*-tree $T_{ad}(G)$, it is clear that $T_{ad}(G)$ consists of those nodes of the tree $T'_c(G)$ which contain available vertices, that is, the construction algorithm establishes a one-to-one correspondence between the nodes V_i of the tree $T'_c(G)$ which contain available vertices and the nodes A_i of the *ad*-tree $T_{ad}(G)$, $1 \leqslant i \leqslant t$. Moreover, the $a\text{-list}(A_i)$ of the nodes A_i contains only the available vertices of the node $V_i \in T'_c(G)$. On the other hand, the $d\text{-list}(A_i)$ may contain dummy vertices from several nodes of the tree $T'_c(G)$, or it may by an empty list.

In Fig. 5, the tree $T'_c(G)$ contains three nodes with available vertices (gray disks), that is, the nodes V_1, V_5 and V_4 with 3, 2 and 2 available vertices, respectively; the *ad*-tree $T_{ad}(G)$ consists of the nodes A_1, A_2 and A_3 which correspond to nodes V_1, V_5 and V_4; the *a*-lists of the nodes A_1, A_2 and A_3 contain the available vertices of the nodes V_1, V_5 and V_4, respectively. The $d\text{-list}(A_1)$ of the tree $T_{ad}(G)$ contains the dummy vertices (white disks) of the nodes V_2 and V_6, while the $d\text{-list}(A_3)$ contains the dummy vertices of the node V_7; the $d\text{-list}(A_2)$ is an empty list.

Observation 5.2. By construction, the trees $T'_c(G)$ and $T_{ad}(G)$ have the following property: if two nodes V_i and V_j have an ancestor (resp. independent) relation in $T'_c(G)$, then the corresponding nodes A_i and A_j have the same relation in the *ad*-tree $T_{ad}(G)$; that is, if $V_i \preccurlyeq V_j$ (resp. $V_i \succcurlyeq V_j$) in the tree $T'_c(G)$, then $A_i \preccurlyeq A_j$ (resp. $A_i \succcurlyeq A_j$) in $T_{ad}(G)$. Recall that, two nodes V_i and V_j are independent if they are not clique-adjacent; see Section 2. It follows that the available vertices of the nodes of the tree $T'_c(G)$ preserve their ancestor relation in the *ad*-tree $T_{ad}(G)$; see Fig. 5: the available vertices of the nodes V_1, V_5 and V_4 of the tree $T'_c(G)$ preserve their ancestor (resp. independent) relation in the tree $T_{ad}(G)$.

The nodes of the tree $T'_c(G)$ which contain dummy and/or black vertices do not appear in the *ad*-tree $T_{ad}(G)$; in Fig. 5, such nodes are: V_2, V_3, V_6 and V_7. Moreover, the black vertices of the tree $T'_c(G)$ do not appear in the *ad*-tree $T_{ad}(G)$. On the other hand, all the dummy vertices of $T'_c(G)$ appear in $T_{ad}(G)$ and have the following property: if an available vertex, say, x, of a node V_i and a dummy vertex, say, y, of a node V_j have an ancestor relation in $T'_c(G)$, then these vertices preserve this relation in the *ad*-tree $T_{ad}(G)$; that is, if $x \in V_i$ and $y \in V_j$, and $V_i \preccurlyeq V_j$ in $T'_c(G)$, then $x \in A_i$ and $y \in A_j$, and $A_i \preccurlyeq A_j$ in the tree $T_{ad}(G)$; the vertices x and y have an ancestor relation if both belong to the same node in $T_{ad}(G)$.

In Fig. 5, the dummy vertices (white disks) of the nodes V_2 and V_6 of the tree $T'_c(G)$ have ancestors the available vertices (gray disks) of the node V_1, while the dummy vertices of the node V_7 have ancestors the available vertices of the nodes V_4 and V_1. Thus, by construction, this relation is preserved in the tree $T_{ad}(G)$; the node A_1 contains the dummy vertices of the nodes V_2 and V_6, and the available vertices of the node V_1, while the node A_3 contains the dummy vertices of the node V_7, and the available vertices of the node V_4.

From the above observations, we state the following properties of the nodes and vertices of the trees $T'_c(G)$ and $T_{ad}(G)$.

Lemma 5.1. *Let $V_1, V_2, \ldots, V_k$ be the nodes of the tree $T'_c(G)$ and let $A_1, A_2, \ldots, V_t$ be the nodes of the ad-tree $T_{ad}(G)$ (see Fig. 5).*

(i) *All the available and dummy vertices of the tree $T'_c(G)$ appears in the ad-tree $T_{ad}(G)$. No black vertex of $T'_c(G)$ appears in $T_{ad}(G)$.*

(ii) *If $x \in V_i$ is an available vertex, $y \in V_j$ is a dummy vertex, and $V_i \preccurlyeq V_j$ in $T'_c(G)$, then $x \in A_i$, $y \in A_j$, and either $A_i \preccurlyeq A_j$ or $A_i = A_j$ in $T_{ad}(G)$.*

(iii) *If $y \in V_j$ is a dummy vertex and $x \in V_i$ is an available vertex, and $V_j \preccurlyeq V_i$ in $T'_c(G)$, then $y \in A_j$ and $x \in A_i$, and $A_j \preccurlyeq A_i$.*

(iv) *If $y \in V_j$ is a dummy vertex and $x \in V_i$ is an available vertex, and $V_j \succcurlyeq V_i$ in $T'_c(G)$, that is, neither $A_j \preccurlyeq A_i$ nor $A_i \preccurlyeq A_j$, then $y \in A_j$ and $x \in A_i$, and either $A_j \preccurlyeq A_i$ or $A_j \succcurlyeq A_i$.*

Proof. It follows directly from the structure of the cent-tree $T_c(G)$ (see Theorem 2.2), the construction of the *ad*-tree $T_{ad}(G)$, that is, Steps (1)–(5), and the Observations 5.1 and 5.2. □

Observation 5.3. In Step (2), all the nodes V_i of the tree $T'_c(G)$ such that $H\text{-label}(V_i) = 0$ are deleted from $T'_c(G)$, $1 \leqslant i \leqslant k$. Thus, if the root V_1 of the tree $T'_c(G)$ has zero H-label, then the result is a forest; note that $H\text{-label}(V_1) \geqslant 0$, for otherwise the input graph G would not be a Hamiltonian graph (see Theorem 3.1). In such a case, we independently work in each tree $T'_c(G_i)$ of the forest, and we assign available vertices of the tree $T'_c(G_i)$ to dummy vertices of the tree $T'_c(G_i)$, $1 \leqslant i \leqslant p_1$, where p_1 is the number of children of V_1. Thus, for simplicity in the description of the algorithm, we assume that the root of the tree $T'_c(G)$ has $H\text{-label}(V_1) > 0$, that is, V_1 contains at least one available vertex, and, thus, V_1 is the root of the *ad*-tree $T_{ad}(G)$.

Having constructed the *ad*-tree $T_{ad}(G)$ of the input graph G, let us now describe an algorithm which assigns available vertices to dummy vertices; more precisely, it

assigns available vertices of the lists a-list(A_i), $1 \leqslant i \leqslant t$, to the dummy vertices of the lists d-list(A_j), $1 \leqslant j \leqslant t$, where A_i is an ancestor of A_j in the ad-tree $T_{ad}(G)$, or $A_i = A_j$.

We assume that the tree $T_{ad}(G)$ is a binary tree rooted at A_1; otherwise, we make it binary using a standard technique [17]. We consider a leaf node, say, A_i, of the tree $T_{ad}(G)$ and let $p(A_i) \neq A_1$ be its parent. Let $p(A_i)$ has two children, that is, the node A_i and the sibling of A_i, denoted by $sib(A_i)$, and let $p(p(A_i))$ be the parent of $p(A_i)$. We apply the *rake* operation at node A_i [17,17a]; that is,

- we remove first the node A_i and, then, the node $p(A_i)$ from the tree $T_{ad}(G)$, and
- we connect the sibling of the node A_i to node $p(p(A_i))$.

Before removing the nodes A_i and $p(A_i)$, we assign available vertices to dummy vertices and update the lists a-list and d-list as follows:

(i) Before the removal of A_i:
Let $a\text{-list}(A_i) = (v_{i1}, v_{i2}, \ldots, v_{ia_i})$ and $d\text{-list}(A_i) = (v'_{i1}, v'_{i2}, \ldots, v'_{id_i})$;
- if $a_i < d_i$, then assign the a_i available vertices of the list a-list(A_i) to dummy vertices $v'_{i1}, v'_{i2}, \ldots, v'_{ia_i}$ of d-list(A_i), and
concatenate the list $(v'_{a_{i+1}}, v'_{a_{i+2}}, \ldots, v'_{id_i})$ to d-list$(p(A_i))$; see Fig. 6;
- if $d_i \leqslant a_i$, then assign the d_i available vertices of the list a-list(A_i) to dummy vertices $v'_{i1}, v'_{i2}, \ldots, v'_{id_i}$ of d-list(A_i); see Fig. 7;
The remaining available vertices $v_{d_{i+1}}, v_{d_{i+2}}, \ldots, v_{ia_i}$ in a-list(A_i) are removed along with the node A_i from the tree $T_{ad}(G)$;

(ii) Before the removal of $p(A_i)$:
Let $a\text{-list}(p(A_i)) = (v_{i1}, v_{i2}, \ldots, v_{ia_i})$ and $d\text{-list}(p(A_i)) = (v'_{i1}, v'_{i2}, \ldots, v'_{id_i})$;
- if $a_i < d_i$, then assign the a_i available vertices of the list a-list$(p(A_i))$ to dummy vertices $v'_{i1}, v'_{i2}, \ldots, v'_{ia_i}$ of d-list$(p(A_i))$, and
concatenate the list $(v'_{a_{i+1}}, v'_{a_{i+2}}, \ldots, v'_{id_i})$ to d-list$(p(p(A_i)))$; see Fig. 6;
- if $d_i \leqslant a_i$, then assign the d_i available vertices of the list a-list$(p(A_i))$ to dummy vertices $v'_{i1}, v'_{i2}, \ldots, v'_{id_i}$ of d-list$(p(A_i))$, and
concatenate the list $(v_{d_{i+1}}, v_{d_{i+2}}, \ldots, v_{ia_i})$ to a-list$(sib(A_i))$; see Fig. 7;

By the above process, that is, the application of the rake operation at leaf node A_i, we establish an one-to-one correspondence between available vertices and dummy vertices of the nodes A_i and $p(A_i)$, and also we update:

(i) the list d-list$(p(p(A_i)))$ by adding dummy vertices or
(ii) the list a-list$(sib(A_i))$ by adding available vertices.

The update operations (i) and (ii) guarantee that, after the application of the rake operation at node A_i, the ancestor relation between the remaining available and dummy vertices in $T_{ad}(G)$ is preserved (see Lemma 5.1). Thus, if we repeatedly apply the rake operations on the tree $T_{ad}(G)$, the available vertices is correctly assigned to dummy vertices, that is, an available vertex v of the node $V_i \in T_c(G)$ is always assigned to a dummy vertex v' such that $v' \in V'_i$ and V_i is an ancestor of V'_i in the cent-tree $T_c(G)$. Moreover, it is known that we can contract the tree $T_{ad}(G)$ into a three-node binary tree, using the rake operation [17]; recall that we have assumed that the

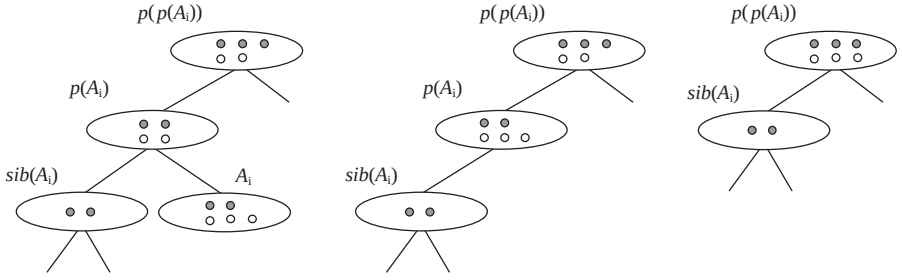

Fig. 6. A rake operation at the node $A_i \in T_{ad}(G)$; it contains two available vertices (gray disks) and three dummy vertices (white disks).

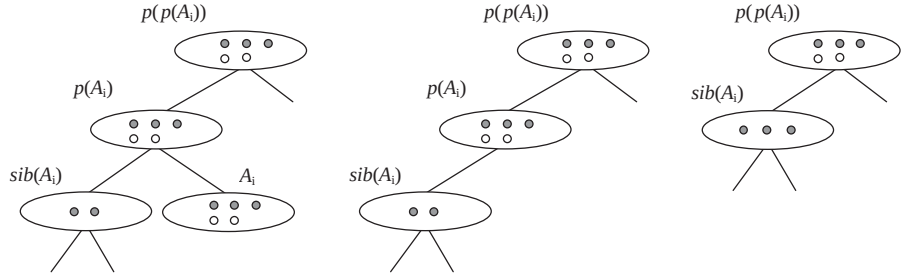

Fig. 7. A rake operation at the node $A_i \in T_{ad}(G)$; it contains three available vertices (gray disks) and two dummy vertices (white disks).

$T_{ad}(G)$ is a binary tree. Let A_2 and A_3 be the children of the root A_1 of the resulting three-node binary tree. Then, we assign available vertices of the root V_1 to dummy vertices of the nodes V_2 and V_3, and complete the assignment process. Note that, since the input graph G is a Hamiltonian graph, the above described algorithm always succeeds in assigning available vertices to all the dummy vertices of the tree $T_{ad}(G)$.

We are interested in implementing the above algorithm in a parallel model of computation. It is known that a binary tree can be efficiently contracted into a three-node tree on the EREW PRAM model using the rake operation [17]. We make the (general) tree *ad*-tree $T_{ad}(G)$ binary as follows: Each node V_i with more that two children is replaced by a balanced binary tree, rooted at V_i, whose leaves are the children of V_i.

Before describing the above algorithm more formally, let us define the notation and the terminology we shall use hereafter. Let $x\text{-list} = (x_1, x_2, \ldots, x_n)$ be a list on n vertices. The position of a vertex x_i in x-list, denoted $\text{pos}(x_i)$, is defined as follows: $\text{pos}(x_1) := 1$ and $\text{pos}(x_i) := \text{pos}(x_{i-1}) + 1$; note that, if the x-list is implemented by an array A, then the $\text{pos}(x_i)$ is the index of x_i in A; if it is implemented by a link-list L, then the $\text{pos}(x_i)$ is the rank of x_i in L. A sublist $(x_i, x_{i+1}, \ldots, x_k)$ of the list x-list is defined to be the list which results from x-list after deleting the vertices with positions less than $\text{pos}(x_i)$ and greater than $\text{pos}(x_k)$.

Let $y\text{-list} = (y_1, y_2, \ldots, y_n)$ be a list on n vertices. By $(y_1, y_2, \ldots, y_n) \leftrightarrow (x_1, x_2, \ldots, x_n)$ denote the one-to-one correspondence between the vertex y_i of the list y-list and the vertex x_i of the list x-list, $1 \leqslant i \leqslant n$. Finally, by x-list $\|$ y-list denote the list $(x_1, x_2, \ldots, x_n, y_1, \ldots, y_n)$, and by $()$ denote the empty list.

We now formally describe the above procedure, which assigns available vertices to dummy vertices, as follows:

Step (B): Assignment of available vertices to dummy vertices

(6) Make the *ad*-tree $T_{ad}(G)$ binary: each node A_i with more that two children is replaced by a balanced binary tree whose leaves are the children of A_i, $1 \leqslant i \leqslant t$; let $A_1, A_2, \ldots, A_{t'}$ be the nodes of the resulting binary tree, $t' \geqslant t$;

(7) For each new internal node A_i of the binary tree $T_{ad}(G)$, compute two vertex lists, namely a-list and d-list, such that $a\text{-list}(A_i) = ()$ and $d\text{-list}(A_i) = ()$;

(8) Compute the position (i.e., index, or rank) of each vertex of the $a\text{-list}(A_i) = (v_{i1}, v_{i2}, \ldots, v_{ia_i})$ and $d\text{-list}(A_i) = (v'_{i1}, v'_{i2}, \ldots, v'_{id_i})$ of the node A_i, $1 \leqslant i \leqslant t'$;

(9) Contract the binary tree $T_{ad}(G)$ into a three-node binary tree, using the *rake* operation;
when a node A_i is subject to rake operation, adjust the a-list and d-list of the nodes A_i, $p(A_i)$, $p(p(A_i))$ and $sib(A_i)$, as follows:

(i) Let $a\text{-list}(A_i) = (v_{i1}, v_{i2}, \ldots, v_{ia_i})$ and $d\text{-list}(A_i) = (v'_{i1}, v'_{i2}, \ldots, v'_{id_i})$;
if $d_i \leqslant a_i$ then
$(v'_{11}, v'_{12}, \ldots, v'_{1d_i}) \leftrightarrow (v_{11}, v_{12}, \ldots, v_{1d_i})$,
else
$(v'_{11}, v'_{12}, \ldots, v'_{1a_i}) \leftrightarrow (v_{11}, v_{12}, \ldots, v_{1a_i})$;
$d\text{-list}(p(A_i)) \leftarrow d\text{-list}(p(A_i)) \| (v'_{1a_{i+1}}, \ldots, v'_{1d_i})$;

(ii) Let $a\text{-list}(p(A_i)) = (v_{i1}, v_{i2}, \ldots, v_{ia_i})$ and $d\text{-list}(p(A_i)) = (v'_{i1}, v'_{i2}, \ldots, v'_{id_i})$;
if $d_i \leqslant a_i$ then
$(v'_{11}, v'_{12}, \ldots, v'_{1d_i}) \leftrightarrow (v_{11}, v_{12}, \ldots, v_{1d_i})$,
$a\text{-list}(sib(A_i)) \leftarrow a\text{-list}(sib(A_i)) \| (v_{1d_{i+1}}, \ldots, v_{1a_i})$
else
$(v'_{11}, v'_{12}, \ldots, v'_{1a_i}) \leftrightarrow (v_{11}, v_{12}, \ldots, v_{1a_i})$;
$d\text{-list}(p(p(A_i))) \leftarrow d\text{-list}(p(p(A_i))) \| (v'_{1a_{i+1}}, \ldots, v'_{1d_i})$;

(10) Let V_1 be the root of the resulting three-node tree and let V_2, V_3 be the children of V_1; Let $a\text{-list}(A_1) = (v_{11}, v_{12}, \ldots, v_{1a_1})$ be the list of the available vertices of V_1;

(i) if $d\text{-list}(A_2)$ contains dummy vertices, say, $v'_{21}, v'_{22}, \ldots, v'_{2d_2}$, then
$(v'_{21}, v'_{22}, \ldots, v'_{2d_2}) \leftrightarrow (v_{11}, v_{12}, \ldots, v_{1d_2})$;

(ii) if $d\text{-list}(A_3)$ contains dummy vertices, say, $v'_{31}, v'_{32}, \ldots, v'_{3d_3}$, then
$(v'_{31}, v'_{32}, \ldots, v'_{3d_3}) \leftrightarrow (v_{1(d_2+1)}, v_{1(d_2+2)}, \ldots, v_{1(d_2+d_3)})$;

Observation 5.4. In Step (10), we assign $d_2 + d_3$ available vertices from the list $a\text{-list}(A_1)$ of the root A_1 of the three-node tree to the dummy vertices of the lists $d\text{-list}(A_2)$ and $d\text{-list}(A_3)$. It is clear that, we can always do this assignment since the input QT-graph G is a Hamiltonian graph, and, thus, if the nodes V_2 and V_3 of the three-node tree contain d_2 and d_3 dummy vertices, respectively, then the root node V_1 contains $a_1 \geqslant d_2 + d_3$ available vertices.

Finally, we show how we can construct the h-tree of input Hamiltonian QT-graph G, given the available-dummy vertex assignment computed in Steps (1)–(10). We proceed as follows:

Let $v'_{i1}, v'_{i2}, \ldots, v'_{in_i}$ be the dummy vertices of the node V'_i of the cent-tree $T_c(G)$, $1 \leqslant i \leqslant p$; see Step (1), and let $v_{i1}, v_{i2}, \ldots, v_{in_i}$ be the corresponding available vertices computed by the above algorithm in Steps (9)–(10). If the vertex v_{ij} belongs to the node $V_j \in T_c(G)$, then we add the vertex v_{ij} to node V'_i and delete it from the node V_j,

$1 \leqslant j \leqslant n_i$. The resulting nodes are the nodes of the h-tree $T_h(G)$ of the input graph G.

The formal description of the last step of the construction algorithm, that is, the step which computes the vertices of each node $V_i \in T_h(G)$, is as follows:

Step (C): *Computation of the vertices of each node of the h-tree*

(11) Let $(v'_{i1}, v'_{i2}, \ldots, v'_{in_i}) \leftrightarrow (v_{i1}, v_{i2}, \ldots, v_{in_i})$, and let $v'_{i1}, v'_{i2}, \ldots, v'_{in_i}$ be the dummy vertices of the node $V'_i \in T_c(G)$, $1 \leqslant i \leqslant p$. Then,
add the vertices $v_{i1}, v_{i2}, \ldots, v_{in_i}$ to the node $V'_i \in T_c(G)$, and
if $v_{ij} \in V_j$, then delete v_{ij} from the node $V_j \in T_c(G)$, $1 \leqslant j \leqslant n_i$.

We call H-TREE-CON the above described algorithm, that is, Steps (1)–(11), which constructs an h-tree $T_h(G)$ of a Hamiltonian QT-graph G, and show that it can be efficiently implemented on the CREW PRAM model of computation.

Correctness. The Steps (1)–(5) of the algorithm H-TREE-CON construct the ad-tree $T_{ad}(G)$ having the properties of Lemma 5.1. The correctness of Steps (6)–(9) follows from Lemma 5.1 and the way we assign available-dummy vertices and update the a-lists and d-lists during each rake operation on the binarized ad-tree $T_{ad}(G)$. Step (10) assigns available vertices from the root A_1 of the resulting three-node tree to the dummy vertices of its two children A_2 and A_3. This assignment always is completed since the input QT-graph is a Hamiltonian graph; see Observation 5.4 and Theorem 3.1. Step (11) completes the construction of the h-tree $T_h(G)$. Thus, the correctness of the algorithm ensues from the correctness of its steps.

Time and processor complexity. We use a step-by-step analysis and compute the time and the number of processors required for the execution of each step of the algorithm H-TREE-CON, and we also determine the type of the PRAM model on which each step is executed.

Step 1: The cent-tree $T_c(G)$ of a QT-graph on n vertices and m edges is constructed in $O(\log n)$ time using $O(n+m)$ processors on the CREW PRAM model; see Algorithm CENT-TREE-CON.

The number of children $V_{i1}, V_{i2}, \ldots, V_{ip_i}$ of a node $V_i \in T_c(G)$, $1 \leqslant i \leqslant k$, can be computed in $O(\log p_i)$ time with $O(p_i/\log p_i)$ processors on the EREW PRAM. It follows that the H-labels of the nodes of the cent-tree $T_c(G)$ are computed in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model. The total number of dummy vertices we add to the nodes $V'_1, V'_2, \ldots, V'_{k'}$ is bounded by $\sum_{i=1}^{k'} (V'_i - p_i + 1) < k = O(n)$. Thus, the nodes of the tree $T'_c(G)$ can be computed in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model. Thus, in total, the execution of the step takes $O(\log n)$ time and requires $O(n+m)$ processors on the CREW PRAM model.

Step 2: Here, the nodes V_i such that H-label$(V_i) = 0$ are deleted from $T'_c(G)$. This can by done using the well-known pointer jumping technique on $T'_c(G)$: it takes $O(\log n)$ time and requires $O(n)$ processors on the CREW PRAM model [17]. It is easy to see that the black vertices can be deleted from the remaining nodes of $T'_c(G)$ in $O(\log n)$ time using $O(n/\log n)$ processors, and, thus, the whole step is executed in $O(\log n)$ time using $O(n+m)$ processors on the CREW PRAM model.

Step 3: The lowest a-node ancestor V_j of the node V_i in $T'_c(G)$ (if the parent $p(V_i)$ of the node V_i is a d-node) can be determined using the pointer jumping technique on $T'_c(G)$. Thus, the step takes $O(\log n)$ time and requires $O(n)$ processors on the CREW PRAM model.

Step 4: In this step, for each a-node $V_i \in T'_c(G)$ two lists are computed: a-list(V_i) contains the available vertices of V_i and d-list(V_i) contains the dummy vertices of all the d-node children of V_i. The total number of available vertices in the nodes of the tree $T'_c(G)$, and, thus, the total number of dummy vertices, is less than n. For the computation of the d-list(V_i) we use an array of size $\sum_{i=1}^{p_i} |V_{ij}|$ and store the vertices of all the children of V_i; then we can select the vertices of the d-node children of V_i using the array packing technique [2]. Thus, this step can be executed in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model.

Step 5: The d-nodes, which are deleted from $T'_c(G)$, are all leaves of the tree $T'_c(G)$. Thus, all the deletion operations can be done in $O(1)$ parallel time with $O(n)$ processors on the EREW PRAM model.

Steps 6 *and* 7: In Step 6, the ad-tree $T_{ad}(G)$ is made binary, while in Step 7 the a-list and d-list of each new internal node of the binary tree $T_{ad}(G)$ are computed to be empty. It is known that an arbitrary n-node tree can be made binary in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model [17]. The a-list and d-list can be obviously computed in $O(1)$ parallel time with $O(n)$ processors on the EREW PRAM model. Therefore, both steps are executed in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model.

Step 8: The a-list(A_i) and the d-list(A_i) of each node A_i of the binary tree $T_{ad}(G)$ are implemented by link-lists, $1 \leqslant i \leqslant t'$. It is well-known that the list-ranking problem on a linked-list with n nodes can be optimally solved in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model [2,17,25]. It is easy to see that, the number of available vertices in the binary tree $T_{ad}(G)$ equals the number of available vertices in the tree $T'_c(G)$; thus, in total, $T_{ad}(G)$ contains less than n available vertices and less than n dummy vertices. It follows that the whole step can be executed in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model.

Step 9: It is well-known that a binary tree can be contracted into a three-node binary tree in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model; the *rake* operation is applied concurrently to several leaves (see [17,25]). When a node A_i of the binary tree $T_{ad}(G)$ is subject to rake operation, the following assignment and update operations are performed:

- Available vertices of the lists a-list(A_i) and a-list$p((A_i))$ are assigned to dummy vertices of the lists d-list(A_i) and d-list$p((A_i))$. Since the rank of each vertex of the lists a-list and d-list is known, an assignment operation is performed in $O(1)$ parallel time with p processors on the EREW PRAM model, where $p = \max\{\ell_a, \ell_d\}$, and ℓ_a, ℓ_b are the lengths of the lists a-list and d-list, respectively.
- The d-list of the node $p(A_i)$, and either the d-list of the node $p(p(A_i))$ or the a-list of the node $sib(A_i)$, are updated using the concatenation operation; for each list, this operation can be done in $O(1)$ sequential time. Then, the rank of each vertex of the updated a-list and d-list is computed. It is easy to see that the list-ranking problem on a list, say, L, which is produced by $L \leftarrow L_1||L_2$, where L_1 and L_2 are two list of lengths ℓ_1 and ℓ_2, respectively, can be solved in O(1) parallel time with ℓ_2 processors on the EREW PRAM model, if the rank of each vertex of the two concatenated lists L_1 and L_2, is known.

Thus, the whole step is executed in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model of computation.

Step 10: This step performs only assignment operations: first, available vertices of the a-list(A_1) are assigned to dummy vertices of the d-list(A_2), and, then, available vertices of the a-list(A_1) are assigned to dummy vertices of the d-list(A_3). Since the rank of each vertex of the lists a-list(A_1), d-list(A_2) and d-list(A_3) is known, this step takes $O(1)$ parallel time with $\max\{\ell_1, \ell_2\}$ processors on the EREW PRAM model, where ℓ_1, ℓ_2 are the lengths of the lists d-list(A_2) and d-list(A_3), respectively.

From the previous step-by-step analysis, it follows that the parallel algorithm H-TREE-CON runs in $O(\log n)$ time using a total of $O(n+m)$ processors on the CREW PRAM model of computation. Thus, we have proved the following result.

Lemma 5.2. *An h-tree $T_h(G)$ of a Hamiltonian QT-graph G on n vertices and m edges can be constructed in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model.*

5.2. Finding of a Hamiltonian cycle

In Section 3, we proved necessary and sufficient conditions for a QT-graph to contain a Hamiltonian cycle; see Theorem 3.1. Based on these conditions, we develop here a parallel algorithm for finding a Hamiltonian cycle in a Hamiltonian QT-graph.

Let G be Hamiltonian QT-graph and let $V_1, V_2, \ldots, V_k$ be the nodes of its cent-tree $T_c(G)$ rooted at V_1. If H-label$(V_i) \geqslant 0$ for each node $V_i \in T_c(G)$, then $T_h(G) := T_c(G)$; otherwise, we construct an h-tree $T_c(G)$ of G from the cent-tree $T_c(G)$. Consider the h-sequence $(V_{f(1)}, V_{a(1)}, \ldots, V_{a(t)} = V_1)$ of the tree $T_h(G)$ and construct the h-dfs tree of the vertices of the graph G using the h-traversal on the tree $T_h(G)$. We select an arbitrary vertex v from the set $V_{f(1)}$ as start point. Since H-label$(V_i) \geqslant 0$ for each node $V_i \in T_h(G)$, it is easy to see that each node of the h-dfs tree rooted at $v \in V_{f(1)}$ has at most one child; its unique leaf u belongs to node V_1 and, thus, $(v,u) \in E(G)$; see Fig. 3. Thus, we can find a Hamiltonian cycle of the graph G from its h-dfs tree.

We have already described efficient parallel algorithms for constructing the cent-tree and the h-tree of a QT-graph G (see Sections 4 and 5.1). Moreover, it is easy to see that the h-sequence of the graph G can be also efficiently constructed in parallel; the leaves $V_{f(1)}, V_{f(2)}, \ldots, V_{f(t)}$ of the cent-tree or the h-tree of G are computed by using the Euler-tour technique and the nodes $V_{a(1)}, V_{a(2)}, \ldots, V_{a(t-1)}$ are computed by solving the LCA problem [2,17,25]. Therefore, it is becoming obvious that we need an efficient parallel algorithm for the h-traversal, that is, a parallel algorithm for constructing the h-dfs tree of the QT-graph G. Thus, we will restrict our attention to design such an algorithm. Note that, no efficient parallel algorithm has been so far developed for the (standard) dfs traversal; various graph numberings, including depth first search, where the numbering algorithm is restricted to a particular order of visiting the edges of the graph, are shown to be P-complete [25].

Next, we describe a method for constructing the h-dfs tree of a Hamiltonian QT-graph G, which leads to an efficient parallel algorithm for constructing a Hamiltonian cycle on the graph G; it works as follows:

(i) We first construct a directed graph F on n vertices, and set $V(F) := V(G)$ and $E(F) := \emptyset$;
(ii) Then, we compute the h-sequence $(V_{f(1)}, V_{a(1)}, \ldots, V_{a(t)} = V_1)$ of the h-tree $T_h(G)$, and the list$(V_i) = (v_{i1}, v_{i2}, \ldots, v_{in_i})$ of each node V_i of the h-tree $T_h(G)$, $1 \leqslant i \leqslant k$; we add the edge $\langle v_{ij}, v_{i(j+1)} \rangle$ in $E(F)$, $1 \leqslant j \leqslant n_i - 1$;
(iii) Let $V_{f(i)}, V_{a(i)}, V_{f(i+1)}$ be three consecutive nodes in the h-sequence, and let $V_{a(i)}$ be the jth occurrence of $V_{a(i)}$ in the h-sequence and let list$(V_{a(i)}) = (v_{i1}, \ldots, v_{ij}, \ldots, v_{in_i})$.

If $i \leqslant t-1$, then we compute the last vertex v_ℓ of the list$(V_{f(i)})$ and the first vertex v_h of the list$(V_{f(i+1)})$; we add the edges $\langle v_\ell, v_{ij} \rangle$ and

$\langle v_{ij}, v_h \rangle$ in $E(F)$, and delete the edge $\langle v_{i(j-1)}, v_{ij} \rangle$ from $E(F)$, $2 \leqslant j \leqslant n_i$.

If $i = t$, then we compute the last vertex v_ℓ of the list$(V_{f(i)})$; we add the edges $\langle v_\ell, v_{ij} \rangle$, and delete the edge $\langle v_{i(j-1)}, v_{ij} \rangle$ from $E(F)$, $2 \leqslant j \leqslant n_i$.

Let v be the first vertex of the node $V_{f(1)}$ and let T be the underline undirected graph of the resulting graph F. By construction, the graph T is a tree. We root the tree T at vertex v and let T_g be the resulting rooted tree. It is easy to see that T_g may contain vertices that have two children (see Fig. 8). Thus, Steps (i)–(iii) of the above method do not guarantee that the h-dfs tree forms a (Hamiltonian) path, that is, an h-dfs path. Fig. 8 depicts the results of Steps (i)–(iii).

Next, we describe how the edge set $E(F)$ can be modified so that the underline undirected graph of the resulting graph F produces a h-dfs path of the graph G. We proceed as follows:

(iv) Let $V_{a(i)}, V_{f(i+1)}$ be two consecutive nodes in the h-sequence $(V_{f(1)}, V_{a(1)}, \ldots, V_{a(t)} = V_1)$, and let list$(V_{a(i)}) = (v_{i1}, \ldots, v_{ij}, \ldots, v_{in_i})$, for $i \leqslant t-1$.

We first determine a vertex v_{ij} of the list$(V_{a(i)})$, $1 \leqslant j \leqslant n_i - 1$, with the following property: $\langle v_{ij}, v_{i(j+1)} \rangle \in E(F)$ and $\langle v_{ij}, v_h \rangle \in E(F)$, where v_h is the first vertex of the list$(V_{f(i+1)})$;

(v) If such a vertex v_{ij} exists, then we delete the edge $\langle v_{ij}, v_h \rangle$ from $E(F)$, determine the last vertex $v_\ell = v_{in_i}$ of the list$(V_{a(i)})$, and add the edge $\langle v_\ell, v_h \rangle$ in $E(F)$;

Let T' be the underline undirected graph of the graph F computed by the above method. Steps (iv)–(v) guarantee that T' is a tree graph. We root the tree graph T' at vertex v and let T'_g be the resulting rooted tree. It is easy to see that each internal vertex of the tree T'_g has now exactly one child, and, thus, T'_g forms a h-dfs path of the graph G (see Fig. 9).

Thus, we can produce a Hamiltonian path $(v = v_0, v_1, \ldots, v_n = u)$ of the input graph G, using the h-dfs tree constructed by the above method. The root v of the h-dfs tree belongs to node $V_{f(1)}$ and its unique leaf u belongs to node V_1 and, thus, $(v, u) \in E(G)$; see Theorem 2.2. Thus, we can extend the Hamiltonian path to a cycle by adding the edge $\langle u, v \rangle$ in $E(F)$ and taking the underline undirected graph, say, HC, of the resulting graph; HC forms a Hamiltonian cycle of G.

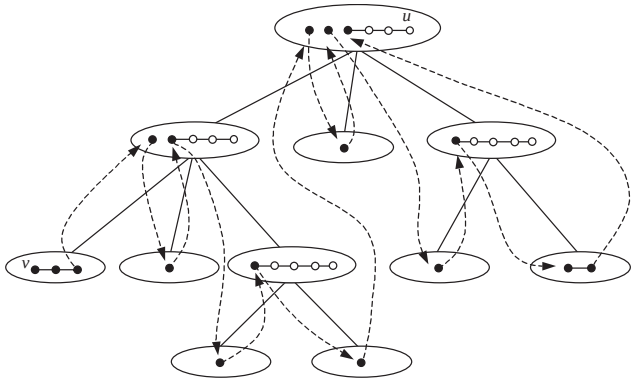

Fig. 8. The results of Steps (i)–(iii) of the h-dfs tree construction of a QT-graph G.

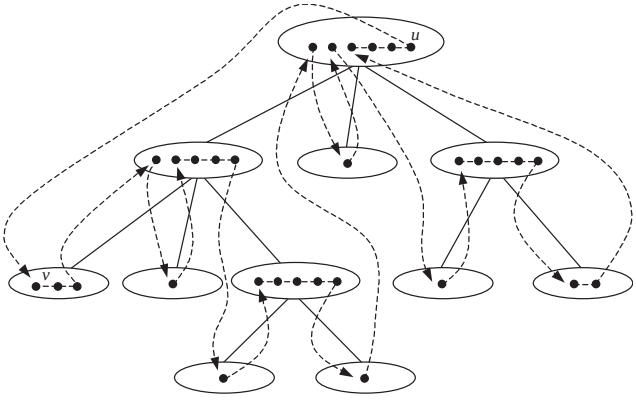

Fig. 9. The structure of a Hamiltonian cycle of a QT-graph G; it is produced using the h-dfs traversal of the h-tree $T_h(G)$.

We next present in a more formal way the parallel algorithm for the construction of a Hamiltonian cycle of a QT-graph; the details of the algorithm are given as follows.

Algorithm Hamiltonian-Cycle-Construction (*HAMILTON-CYCLE-CON*):

Input: a QT-graph G on n vertices and m edges, which is Hamiltonian.

Output: a Hamiltonian cycle HC of the input graph G.

1. Compute the cent-tree $T_c(G)$ of the input graph G; let $V_1, V_2, \ldots, V_k$ be the nodes of the cent-tree $T_c(G)$ rooted at $r_c = V_1$;
2. If H-label$(V_i) \geqslant 0$, for $1 \leqslant i \leqslant k$, then $T_h(G) \leftarrow T_c(G)$; otherwise, compute an h-tree $T_h(G)$ of the input graph G;
3. Compute the h-sequence$(T_h) = (V_{f(1)}, V_{a(1)}, V_{f(2)}, \ldots, V_{f(t)}, V_{a(t)} = V_1)$;
4. Construct a directed graph F, in parallel, as follows:
   $V(F) \leftarrow V(G)$ and $E(F) \leftarrow \emptyset$, and paint its vertices black;
5. For each vertex $V_i \in T_h(G)$, $1 \leqslant i \leqslant k$, do in parallel
   construct the linked-list list$(V_i) = (v_{i1}, v_{i2}, \ldots, v_{in_i})$, and
   add the edge $\langle v_{ij}, v_{i(j+1)} \rangle$ in $E(F)$, $1 \leqslant j \leqslant n_i - 1$;
6. For each internal node $V_{a(i)} \in h$-sequence(T_h), $1 \leqslant i \leqslant t$, do in parallel
   let list$(V_{a(i)}) = (v_{i1}, \ldots, v_{ij}, \ldots, v_{in_i})$; $V_{f(t+1)} \leftarrow V_{f(1)}$;
   if $V_{a(i)}$ is the jth occurrence of $V_{a(i)}$ in the h-sequence(T_h), then

• find the jth vertex v_{ij} of the list$(V_{a(i)})$; paint v_{ij} with read color;
• add $\langle v_\ell, v_{ij} \rangle$ in $E(F)$, where v_ℓ is the last vertex of the list$(V_{f(i)})$;
• add $\langle v_{ij}, v_h \rangle$ in $E(F)$, where v_h is the first vertex of the list$(V_{f(i+1)})$;

7. For each internal node $V_{a(i)} \in h$-sequence(T_h), $1 \leqslant i \leqslant t$, do in parallel
let list$(V_{a(i)}) = (v_{i1}, \ldots, v_{ij}, \ldots, v_{in_i})$; $V_{f(t+1)} \leftarrow V_{f(1)}$;
if v_{ij} and $v_{i(j+1)}$ are read vertices of the list$(V_{a(i)})$, then
• delete edge $\langle v_{ij}, v_{i(j+1)} \rangle$ from the set $E(F)$;
else-if v_{ij} is a read vertex and $v_{i(j+1)}$ is a black vertex, then
• delete $\langle v_{ij}, v_h \rangle$ from $E(F)$, where v_h is the first vertex of the list$(V_{f(i+1)})$;
• add $\langle v_\ell, v_h \rangle$ in $E(F)$, where v_ℓ is the last vertex of the list$(V_{a(i)})$;
8. Construct a spanning cycle HC of G, in parallel, as follows:
set $V(HC) \leftarrow V(F)$;
add (v_i, v_j) in $E(HC)$, for every edge $\langle v_i, v_j \rangle$ in $E(F)$;

Correctness. The correctness of the parallel algorithm HAMILTON-CYCLE-CON is established through Theorem 3.1 and the correctness of the h-tree construction algorithm H-TREE-CON; see Lemma 5.2.

Time and processor complexity. We next compute the time and processor complexity of the proposed parallel algorithm for the construction of a Hamiltonian cycle of a QT-graph G; its step-by-step analysis is as follows.

Step 1: The cent-tree $T_c(G)$ of a QT-graph on n vertices and m edges is constructed in $O(\log n)$ time using $O(n+m)$ processors on the CREW PRAM model; see Algorithm CENT-TREE-CON.

Step 2: The H-labels of the nodes of the cent-tree $T_c(G)$ are computed in $O(\log n)$ time with $O(n/\log n)$ processors on the EREW PRAM model; see Step 1—Algorithm H-TREE-CON. The h-tree $T_h(G)$ of a Hamiltonian QT-graph is constructed in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model; see Lemma 5.1.

Step 3: Let h-sequence$(T_h) = (V_{f(1)}, V_{a(1)}, V_{f(2)}, \ldots, V_{a(t)} = V_1)$. By definition, the node sequence $(V_{f(1)}, V_{f(2)}, \ldots, V_{f(t)})$ is a left-to-right order listing of the leaves of the tree $T_h(G)$, and the node $V_{a(i)}$ is the LCA of the nodes $V_{f(i)}$ and $V_{f(i+1)}$, where $1 \leqslant i \leqslant t-1$. It is well-known that we can optimally compute the left-to-right order listing of the leaves of a tree in $O(\log n)$ parallel time on the EREW PRAM model by using the Euler-tour technique [2,17,25]. We can also optimally compute the LCA of two vertices of a rooted tree in $O(\log n)$ parallel time on the CREW PRAM model [2,17,25]. Thus, the h-sequence of the h-tree $T_h(G)$ can be computed in $O(\log n)$ time with $O(n/\log n)$ processors on the CREW PRAM model.

Step 4: The directed graph F can be constructed in $O(1)$ parallel time with $O(n)$ processors on the EREW PRAM model; its vertices can be painted within the same time and processor bounds.

Step 5: Having computed the vertices of the nodes $V_1, V_2, \ldots, V_k$ of $T_h(G)$, it is easy to see that the linked-lists list$(V_i) = (v_{i1}, v_{i2}, \ldots, v_{in_i})$, $1 \leqslant i \leqslant k$, can be constructed in $O(1)$ parallel time with a total of $O(n)$ processors on the EREW PRAM model; note that $\sum_{i=1}^{k} n_i = n$. Obviously, all the edges $\langle v_{ij}, v_{i(j+1)} \rangle$, for $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant n_i - 1$, can be added in the set $E(F)$ in $O(1)$ parallel time with $O(n)$ processors on the EREW PRAM model.

Step 6: Let list$(V_{a(i)}) = (v_{i1}, \ldots, v_{ij}, \ldots, v_{in_i})$ be the list of the vertices of the node $V_i \in T_h(G)$, $1 \leqslant i \leqslant k$. It is well-known that the list-ranking of list(V_i) determines the distance of each vertex v_{ij} from the first vertex v_{i1} of the list. The list-ranking problem on a list with n_i vertices can be solved in $O(\log n_i)$ time with $O(n_i/\log n_i)$ processors on the EREW PRAM model, $1 \leqslant i \leqslant k$. Thus, we can rank all the lists list(V_i), $1 \leqslant i \leqslant k$, in $O(\log n)$ time with $O(n)$ processors on the EREW PRAM model. Then, we can easily see that all the operations of this step are executed within the same time and processor bounds.

Step 7: Having computed the rank of each vertex of the list list(V_i), $1 \leqslant i \leqslant k$, it is easy to see that, this step can be executed in $O(1)$ time with $O(n)$ processors on the CREW PRAM model.

Step 8: Since the connected directed graph F contains $O(n)$ edges, the spanning cycle HC can be constructed in $O(1)$ parallel time with $O(n)$ processors on the EREW PRAM model.

Therefore, from the previous step-by-step analysis, it follows that the Hamiltonian cycle construction Algorithm HAMILTON-CYCLE-CON runs in $O(\log n)$ time using a total of $O(n+m)$ processors on the CREW PRAM model. Thus, we have proved the following result.

Theorem 5.1. *A Hamiltonian cycle of a QT-graph can be constructed in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model.*

6. Recognition and Hamiltonian completion number

The $O(\log n)$-time parallel algorithm H-TREE-CON for constructing the h-tree of a Hamiltonian QT-graph can be easily modified so that it can also be served as a recognition algorithm for Hamiltonian QT-graphs: if the input QT-graph G is not a Hamiltonian graph, then the algorithm fails to construct the h-tree $T_h(G)$.

It is possible, however, to obtain a much simpler parallel algorithm for recognizing whether the input QT-graph G is a Hamiltonian graph. We describe, first, a simple $O(\log n)$-time parallel recognition algorithm working with a linear number of processors, and, then, we show that the problem of computing the Hamiltonian completion number of a QT-graph can also be solved within the same time and processor bounds.

6.1. Recognition of a Hamiltonian QT-graph

Let G be QT-graph and let $V_1, V_2, \ldots, V_k$ be the nodes of its cent-tree $T_c(G)$ rooted at V_1. Based on Theorem 3.1, we check whether H-label$(V_i) \geqslant 0$ for each node $V_i \in T_c(G)$; if so, then the QT-graph is a Hamiltonian graph; otherwise, we apply the contraction process on the binarized cent-tree $T_c(G)$ and properly count the dummy-available vertices in $T_c(G)$ after each *rake* operation.

More precisely, let H-label(V_i) be the H-label of the node $V_i \in T_c(G)$. It is easy to see that the H-label(V_i) determines the number of the dummy-available vertices of the node V_i: if H-label$(V_i) > 0$, then V_i contains H-label(V_i) available vertices, while if H-label$(V_i) < 0$, it contains $|H\text{-label}(V_i)|$ dummy vertices; see Step (1) of the algorithm H-TREE-CON.

Thus, we assign a number $h(V_i)$ to each node of the binarized cent-tree, say, $T_b(G)$, and initially set $h(V_i) := H$-label(V_i) for each $V_i \in T_b(G)$; we call it *h-number* of the node V_i. If a node V_i of the tree $T_b(G)$ is subject to rake operation, we adjust either the h-number of the node $p(p(V_i))$ or the h-number of the node $sib(V_i)$ as follows:

(i) Before the removal of A_i:
- if $h(V_i) < 0$, then $h(p(V_i)) \leftarrow h(p(V_i)) + h(V_i)$;

(ii) Before the removal of $p(A_i)$:
- if $h(p(V_i)) < 0$, then $h(p(p(V_i))) \leftarrow h(p(p(V_i))) + h(p(V_i))$;
- if $h(p(V_i)) > 0$, then $h(sib(V_i)) \leftarrow h(sib(V_i)) + h(p(V_i))$;

Let V_1 be the root of the resulting three-node tree and let V_{11}, V_{12} be the children of V_1. From the correctness of the algorithm H-TREE-CON, which constructs the h-tree of a Hamiltonian QT-graph G, it follows that if $h(V_i) < 0$ then the node V_i contains $d_i = |h(V_i)|$ dummy vertices and $a_i = 0$ available vertices; otherwise, it contains $d_i = 0$ dummy vertices and $a_i = h(V_i)$ available vertices, $i = 1, 11, 12$. Thus, the h-tree $T_h(G)$ of the graph G can be constructed, and, thus, G is a Hamiltonian graph, if $a_1 \geqslant 0$ and $a_1 \geqslant d_{11} + d_{12}$; in all the other cases, G is not a Hamiltonian graph.

Thus, we can decide whether a QT-graph is a Hamiltonian graph by applying the contraction process on the binarized cent-tree $T_c(G)$ and computing the h-numbers of the nodes V_1, V_{11} and V_{12} of the resulting three-node tree. This computation is described in the following algorithm.

Algorithm Hamiltonian-graph-recognition (*HAMILTON-REC*):

Input: a quasi-threshold graph G on n vertices and m edges.

Output: yes, if G is a Hamiltonian graph; otherwise, no.

1. Compute the cent-tree $T_c(G)$ of the graph G, and make it binary;
   let $T_b(G)$ be the resulting binary tree and let $V_1, V_2, \ldots, V_{k'}$ be its nodes, $k' \geqslant k$;
2. For each node $V_i \in T_b(G)$, $1 \leqslant i \leqslant k'$, compute its H-label(V_i) and set:
   $h(V_i) \leftarrow H$-label(V_i);
3. If $h(V_i) \geqslant 0$, for each node $V_i \in T_b(G)$, $1 \leqslant i \leqslant k'$, then G is a Hamiltonian QT-graph; exit;
4. Contract the binary tree T_b into a three-node binary tree, using the *rake* operation;
   when a node V_i is subject to rake operation, we adjust either the h-number of the node $p(p(V_i))$ or the h-number of the node $sib(V_i)$, as follows:
   if $h(V_i) < 0$ then
   if $h(V_i) + h(p(V_i)) < 0$
   then $h(p(p(V_i))) \leftarrow h(p(p(V_i))) + h(V_i) + h(p(V_i))$
   else
   $h(sib(V_i)) \leftarrow h(sib(V_i)) + h(V_i) + h(p(V_i))$;
   if $h(V_i) \geqslant 0$ then
   if $h(p(V_i)) < 0$
   then $h(p(p(V_i))) \leftarrow h(p(p(V_i))) + h(p(V_i))$
   else $h(sib(V_i)) \leftarrow h(sib(V_i)) + h(p(V_i))$;
5. Check the h-numbers of the nodes V_1, V_{11} and V_{12}, of the three-node binary tree;
   if $h(V_{11}) > 0$ then $h(V_{11}) \leftarrow 0$, and if $h(V_{12}) > 0$ then $h(V_{12}) \leftarrow 0$;
   if $h(V_1) \geqslant 0$ and $h(V_1) \geqslant h(V_{11}) + h(V_{12})$ then G is a Hamiltonian QT-graph; otherwise, G is not a Hamiltonian QT-graph;

From the step-by-step analysis of the algorithm H-TREE-CON for constructing the h-tree of a Hamiltonian QT-graph (see Section 5.1), we can easily conclude that the Steps (1)–(4) of the proposed recognition algorithm are executed in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model. Obviously, the Step 5 is executed in $O(1)$ sequential time. Thus, we have the following result.

Theorem 6.1. *It can be decided whether a QT-graph on n vertices and m edges is a Hamiltonian graph in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model.*

6.2. Hamiltonian completion number

Let G be a nonHamiltonian QT-graph on n vertices and m edge, and let V_1, V_{11} and V_{12} be the nodes of the three-node tree computed by the algorithm Hamilton-Rec; V_1 is the root of the tree and V_{11}, V_{12} are its children.

Since G is not a Hamiltonian graph, at least one of the h-numbers $h(V_1)$, $h(V_{11})$ and $h(V_{12})$ has value less than zero. Recall that, if $h(V_i)<0$ then the node V_i contains $d_i=|h(V_i)|$ dummy vertices and $a_i=0$ available vertices, while if $h(V_i)\geqslant 0$, it contains $d_i=0$ dummy vertices and $a_i=h(V_i)$ available vertices, $i=1,11,12$. Moreover, if the nodes V_{11} and V_{12} contain a total of $d_{11}+d_{12}\geqslant 0$ dummy vertices, then the root node V_1 contains either $d_1<0$ dummy vertices or $a_1<d_{11}+d_{12}$ available vertices; G is not a Hamiltonian graph, and, thus, the h-tree $T_h(G)$ of the graph G cannot be constructed (see Theorem 3.1).

The graph G becomes a Hamiltonian QT-graph if we add $n'=|d_{11}+d_{12}+h(V_1)|$ vertices in the root node V_1 of the cent-tree $T_c(G)$ and a number of appropriate edges in $E(G)$; that is, $n'=d_{11}+d_{12}+d_1$ if $d_1<0$, and $n'=d_{11}+d_{12}-a_1$ if $a_1\geqslant 0$. The correctness of the process for assignment available vertices to dummy vertices, that is, Steps (1)–(10) of the algorithm H-TREE-CON, implies that n' is the minimum number of vertices which need to be added to $V_1\in T_c(G)$, along with m' edges, to make G a Hamiltonian QT-graph, where $m'=n\cdot n'+n'\cdot(n'-1)/2$; recall that V_1 is a clique and $V_1=cent(G)$ (see Section 2). We denote G' the resulting Hamiltonian QT-graph.

More precisely, the graph G' is a Hamiltonian QT-graph on $n+n'$ vertices and $m+m'$ edges; it has vertex set $V(G')=V(G)\cup V'$, where $V'=\{u'_1,u'_2,\ldots,u'_{n'}\}$ and edge set $E(G')=E(G)+E'$, where E' contains m' edges (x,y) such that $x\in V'$ and $y\in V(G)\cup V'$; note that $x\neq y$ since we consider graphs with no loops. The cent-trees $T_c(G')$ and $T_c(G)$ have the same structure, and G' contains the graph G as an induces subgraph.

The Hamiltonian completion number $hcn(G)$ of a graph G is defined to be the minimum number of edges which need to be added to $E(G)$ to make G Hamiltonian [3,13]. We prove the following.

Lemma 6.1. *The number $n'=|d_{11}+d_{12}+h(V_1)|$ equals the Hamiltonian completion number $hcn(G)$ of a non-Hamiltonian QT-graph G.*

Proof. Let $V_1,V_2,\ldots,V_k$ be the nodes of the cent-tree $T_c(G)$ of the QT-graph G rooted at V_1. We add n' vertices, say, $u'_1,u'_2,\ldots,u'_{n'}$, in the root node V_1 and m' edges in $E(G)$, and let G' be the resulting Hamiltonian QT-graph. By construction, the cent-tree $T_c(G')$ has nodes $V'_1,V_2,\ldots,V_k$, where V'_1 is the root of $T_c(G')$ and $V'_1=V_1\cup V'$, and the vertex set $V'=\{u'_1,u'_2,\ldots,u'_{n'}\}$ is minimal.

Consider the h-sequence $(V_{f(1)},V_{a(1)},\ldots,V_{a(t)}=V'_1)$ of the h-tree $T_h(G')$ and the h-dfs tree of the graph G'. Let $v\in V_{f(1)}$ be the root of the h-dfs tree and let $HC'=(v,\ldots,v_i,u',v_j,\ldots,v)$ be the Hamiltonian cycle which is produced by the h-dfs tree, where $u'\in V'$. Note that, HC' is a cycle on $n+n'$ vertices. By construction, the cycle HC' has the following properties:

(i) If $u'\in V'$, then its two adjacency vertices in HC', say, v_i and v_j, are not vertices of V'; that is, $v_i,v_j\in V(G)$, and
(ii) if $v_i\in V_i$ and $v_j\in V_j$, then both nodes V_i and V_j are leaves of the cent-tree $T_c(G')$ and $V_i\neq V_j$.

Property (i) follows from the h-dfs traversal of the cent-tree $T_c(G')$ (see Section 3): if V_1 (note that $V'\subseteq V_1$) is not its last occurrence in the h-sequence, then the h-dfs visits only one unvisited vertex from V_1, while if V_1 is its last occurrence, then V_1 contains exactly one unvisited vertex since the number n' is minimum. The property (ii) follows from the structure of the h-sequence: the internal nodes (including the root node) and the leaf nodes alternate in the h-sequence, and, thus, V_i and V_j are leaves since $v_i\in V_i$, $v_j\in V_j$ and $u'\in V_1$.

Thus, if we remove each vertex $u'\in V'$ from HC' and make the vertices v_i and v_j to be adjacent, the resulting structure HC^* is a cycle on n vertices $v_1,v_2,\ldots,v_n$, where $v_i\in V(G)$. On the other hand, since the vertices v_i and v_j belong to deferent leave nodes of $T_c(G')$, it follows that v_i and v_j are not adjacent in the graph G. Thus, if we add n' edges of the form (v_i,v_j) in $E(G)$, then the resulting graph is a Hamiltonian graph and the cycle HC^* is a Hamiltonian cycle of it.

The vertices $v_i\in V_i$ and $v_j\in V_j$ belong to leaf nodes, that is, $(v_i,v_j)\notin E(G)$, and the vertex $u'\in V'$ cannot be replaced by a vertex $v\in V(G)$, for otherwise V' would contain less that n' vertices, in contradiction to the fact that V' is minimal. It follows that, the number n' of edges of the form (v_i,v_j) that need to be added to $E(G)$ to make G Hamiltonian is minimum. Thus, $n'=hcn(G)$ and the lemma is proved. □

By combining the Lemma 6.1 with the recognition algorithm Hamilton-Rec, we obtain the following parallel algorithm for computing the Hamiltonian completion number $hcn(G)$ of the QT-graph; it takes as input a nonHamiltonian QT-graph G on n vertices and m edges.

Algorithm Hamiltonian-Completion-Number (*HC-NUMBER*):

1. Execute the Steps (1)–(4) of the Hamiltonian QT-graph recognition algorithm;

2. Compute the Hamiltonian completion number $hcn(G)$ of G as follows:
   if $h(V_{11})>0$ then $h(V_{11})\leftarrow 0$, and if $h(V_{12})>0$ then $h(V_{12})\leftarrow 0$;
   $hcn(G)\leftarrow |h(V_1)+h(V_{11})+h(V_{12})|$;

The time and the processor complexity of the proposed algorithm *HC-NUMBER* can be easily computed: Step 1 is executed in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model (see algorithm HAMILTON-REC), while Step 2 is executed in $O(1)$ sequential time. Thus, we state the following result.

Theorem 6.2. *The Hamiltonian completion number of a QT-graph on n vertices and m edges can be computed in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model.*

7. Coloring and other optimization problems

The algorithmic properties of the QT-graphs, which we have shown in this paper, allow us to efficiently solve other optimization problems on such graphs in parallel. Specifically, we can solve the coloring problem, the maximum clique problem, the maximum independent set problem and other problems in $O(\log n)$ time using a linear number of processors on the CREW PRAM model.

Let G be a QT-graph and let $V_1, V_2, \ldots, V_k$ be the nodes of the cent-tree $T_c(G)$. We have shown, in Section 2, that for every two nodes V_s and V_t such that $V_s \preccurlyeq V_t$; that is, V_s is an ancestor of V_t in $T_c(G)$, the graph $G[\{\bigcup V_i \mid V_s \preccurlyeq V_i \preccurlyeq V_t\}]$ is a complete graph. Moreover, for every maximal element V_t of $(V_i, \preccurlyeq)$, the graph $G[\{\bigcup V_i \mid V_1 \preccurlyeq V_i \preccurlyeq V_t\}]$ is a maximal complete subgraph of G; see also Observation 2.1.

Based on these properties, it is easy to see that the height of the tree $T_d(G)$ (see Section 4), equals the clique number $\omega(G)$ minus 1; recall that the height of a vertex of $T_d(G)$ is the number of edges in the longest path from the vertex in question to a leaf—all the leaves of $T_d(G)$ have 0 heights. Moreover, the set which contains the vertices of the ith level of the tree $T_d(G)$ is a stable set. Since $\omega(G)=\chi(G)$, we can color the graph G by computing the level $\ell(v_i)$ of each vertex v_i of the tree $T_d(G)$ and setting $color(v_i) := \ell(v_i)$, $1\leqslant i\leqslant n$; assuming that $\ell(r)=1$, where r is the root of $T_d(G)$.

Let u be a leaf of the degree-tree $T_d(G)$ such that $\ell(u)=\omega(G)$ and let MC be the set of vertices of the path from the root r of $T_d(G)$ to vertex u. Then, the vertex set MC is the maximum clique of the graph G. Thus, we can easily compute the set MC using the parallel pointer jumping technique on the tree $T_d(G)$.

Let $S=\{v_s, v_{s+1}, \ldots, v_t, \ldots, v_q\}$ be a stable set such that $v_t \in V_t$ and V_t is a maximal element of $(V_i, \preccurlyeq)$ or, equivalently, V_t is a leaf node of cent-tree $T_c(G)$, $s\leqslant t\leqslant q$. It is easy to see that S has the maximum cardinality $\alpha(G)$ among all the stable sets of G. It is also easy to see that the set S contains the leave vertices of the tree $T_d(G)$. Recall that, the leaves of a tree can be found using the Euler-tour technique [17].

Taking into consideration the above discussion, the complexity of the algorithms for constructing the trees $T_d(G)$ and $T_c(G)$ (see Section 4), and the complexity of some standard algorithmic techniques for computing the level function, the set of the leaves and certain paths on the degree-tree and cent-tree of G [2,17,25], we state the following results.

Theorem 8.1. *The problems of coloring a QT-graph G on n vertices and m edges and finding the maximum clique and the maximum independent set of G can be solved in $O(\log n)$ time with $O(n+m)$ processors on the CREW PRAM model.*

8. Concluding remarks

In this paper we showed structural and algorithmic properties on the class of QT-graphs and proved necessary and sufficient conditions for a QT-graph to be Hamiltonian. We also showed that a QT-graph G has a unique tree representation, that is, the cent-tree $T_c(G)$, which meets the structural properties of G.

By taking advantage of these properties and conditions, we presented efficient parallel algorithms for constructing the cent-tree $T_c(G)$ and finding a Hamiltonian cycle of a QT-graph G; our algorithms run in $O(\log n)$ time and require $O(n+m)$ processors on the CREW PRAM model, where n is the number of vertices and m is the number of edges of the input graph. In addition, we presented a simple $O(\log n)$-time parallel algorithm for recognizing whether a QT-graph is Hamiltonian which requires $O(n+m)$ processors, and we showed that the problem of computing the Hamiltonian completion number of a QT-graph can also be solved in $O(\log n)$ time with $O(n+m)$ processors. We also presented parallel algorithms for other optimization problems on QT-graphs which run in $O(\log n)$ time using a linear number of processors.

Different problems can be foreseen for further research. An interesting optimization problem is the construction of a Hamiltonian cycle of a QT-graph G in the weighted case: each vertex and/or edge of G has certain weight and we wish to minimize the total weight of edges in a Hamiltonian cycle (for results on "heavy" paths and cycles in weighted graphs; see [30]). A second problem that is worth studying is the weighted version of the Hamiltonian completion problem: we wish to minimize the total weight of the edges which need to be

added to $E(G)$ to make G Hamiltonian. We pose these as open problems for algorithmic study.

A topic for further research is the study of problems on the line graph of a QT-graph (for results on line graphs; see [6,28]). One can work towards the identification of structural and algorithmic properties of such graphs, which may lead to parallel and/or sequential algorithms for the Hamiltonian problems we consider here, as well as for other combinatorial and optimization problems.

References

[1] G. Adhar, S. Peng, Parallel algorithms for cographs and parity graphs with applications, J. Algorithms 11 (1990) 252–284.
[2] S.G. Akl, Parallel Computation: Models and Methods, Prentice-Hall, Englewood Cliffs, NJ, 1997.
[3] A.A. Bertossi, The edge Hamiltonian problem is NP-hard, Inform. Process. Lett. 13 (1981) 159–177.
[4] H.L. Bodlaender, R.H. Möhring, The pathwidth and treewidth of cographs, SIAM J. Discrete Math. 6 (1993) 181–188.
[5] A. Brandstadt, B. Le, J.P. Spinrad, Graph classes: a survey, SIAM Monographs on Discrete Mathematics and its Applications, SIAM, Philadelphia, PA, 1999.
[6] R.A. Brualdi, R.F. Shong, Hamiltonian line graphs, J. Graph Theory 5 (1981) 304–307.
[7] V. Chvátal, P.L. Hammer, Set-packing and threshold graphs, Research Report CORR 73-21, University of Warerloo, 1973.
[8] D.G Corneil, H. Lerches, L. Burlingham, Complement reducible graphs, Discrete Appl. Math. 3 (1981) 163–174.
[9] D.G. Corneil, Y. Perl, L.K. Stewart, A linear recognition algorithm for cographs, SIAM J. Comput. 14 (1985) 926–934.
[10] E. Dahlhaus, Efficient parallel recognition algorithms of cographs and distance hereditary graphs, Discrete Appl. Math. 57 (1995) 29–44.
[11] M.C. Golumbic, Trivially perfect graphs, Discrete Math. 24 (1978) 105–107.
[12] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.
[13] S. Goodman, S. Hedetniemi, On the Hamiltonian completion problems, in: A. Dold, B. Eckman (Eds.), Graphs and Combinatorics, Lecture Notes in Mathematics, Vol. 406, Springer, Berlin, 1974, pp. 262–272.
[14] F. Harary, Graph Theory, Addison–Wesley, Reading, MA, 1969.
[15] F. Harary, S.D. Nikolopoulos, On complete systems of invariants for small graphs, Internat. J. Comput. Math. 64 (1997) 35–46.
[16] X. He, Parallel algorithms for cographs recognition with applications, J. Algorithms 15 (1993) 284–313.
[17] J. JáJá, An Introduction to Parallel Algorithms, Addison-Wesley, Reading, MA, 1992.
[17a] S.R. Kosaraju, A. Delcher, Optimal parallel evaluation of tree-structured computations by ranking, in Proceedings of AWOC'88, Springer-Verlag, 1988, pp. 101–110.
[18] H. Lerchs, On cliques and kernels, Department of Computer Science, University of Toronto, March 1971.
[19] R. Lin, S. Olariu, An NC recognition algorithm for cographs, J. Parallel Distrib. Comput. 13 (1991) 76–90.
[19a] R. Lin, S. Olariu, An optimal parallel matching algorithm for cographs, J. Parallel Distrib. Comput. 22 (1994) 26–36.
[20] R. Lin, S. Olariu, G. Pruesse, An optimal path cover algorithm for cographs, Computers Math. Appl. 30 (1995) 75–83.
[21] S. Ma, W.D. Wallis, J. Wu, Optimization problems on quasi-threshold graphs, J. Comb. Inform. System Sci. 14 (1989) 105–110.
[22] K. Nakano, S. Olariu, A. Zomaya, A time-optimal solution for the path cover problem on cographs, Theoret. Comput. Sci. 290 (2003) 1541–1556.
[23] S.D. Nikolopoulos, Constant-time parallel recognition of split graphs, Inform. Process. Lett. 54 (1995) 1–8.
[24] S.D. Nikolopoulos, Recognizing cographs and threshold graphs through a classification of their edges, Inform. Process. Lett. 75 (2000) 129–139.
[25] J. Reif (Ed.), Synthesis of Parallel Algorithms, Morgan Kaufmann, Los Altos, CA, 1993.
[26] E.S. Wolk, The comparability graph of a tree, Proc. Amer. Math. Soc. 3 (1962) 789–795.
[27] E.S. Wolk, A note of the comparability graph of a tree, Proc. Amer. Math. Soc. 16 (1965) 17–20.
[28] H.J. Veldman, A result on Hamiltonian line graphs involving restrictions on induced subgraphs, J. Graph Theory 12 (1988) 413–420.
[29] J-H. Yan, J-J. Chen, G.J. Chang, Quasi-threshold graphs, Discrete Appl. Math. 69 (1996) 147–255.
[30] S. Zhang, X. Li, H.J. Broersma, Heavy paths and cycles in weighted graphs, Discrete Math. 223 (2000) 327–336.

Stavros D. Nikolopoulos received the B.Sc. degree (with honours) in mathematics from the University of Ioannina, Greece, in 1982, the M.Sc degree in computer science from University of Dundee, Scotland, in 1985, and the Ph.D. degree in computer science from the University of Ioannina, in 1991. He was a research assistant in the Saclant Undersea Research Centre, Italy, in 1990-91. From 1992 to 1996 he was a lecturer in the Department of Computer Science, University of Cyprus, Nicosia. In 1996 he joined the Department of Computer Science, University of Ioannina, Greece, where he is currently an associate professor. His research interests focus on parallel algorithms, data structures and algorithms, graph theory, graph modelling, combinatorial and graph algorithms, combinatorial mathematics, simulation techniques, discrete event systems.