PARALLEL ALGORITHMS FOR HAMILTONIAN PROBLEMS ON QUASI-THRESHOLD GRAPS

S.D. Nikolopoulos

3– 2002

Preprint, no 3 – 02 / 2002

Department of Computer Science
University of Ioannina
45110 Ioannina, Greece

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
email: stavros@cs.uoi.gr

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 efficient parallel algorithms for finding a Hamiltonian cycle and computing the Hamiltonian completion number and a Hamiltonian completion edge set of a *QT*-graph; for an input graph on n vertices and m edges, our algorithms take $O(\log^2 n)$ time and require $O(n + m)$ processors on the CREW PRAM model. In addition, we show 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. This implies an $O(\log n)$-time parallel algorithm for recognizing whether a *QT*-graph is a Hamiltonian graph. 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; the set of such edges is called *Hamiltonian completion edge set* of G [5, 14]. We denote the Hamiltonian completion number of a graph G as $hcn(G)$ and its Hamiltonian completion edge set as $CE(G)$. If G is a Hamiltonian graph, then $hcn(G) = 0$.

Given a graph G, an edge $(x, y) = (y, x)$ of G can be classified as follows according to the relationship of closed neighbourhoods [18, 25, 26]: (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. Obviously, $E(G)$ can be partitioned into the three subsets of free edges, semi-free edges and of actual edges, respectively.

A graph G is called a *quasi-threshold graph*, or QT-graph for short, if every edge of G is either free or semi-free. Thus G is a QT-graph if and only if for every edge (x, y) of G, we have $N[x] \subseteq N[y]$ or $N[x] \supseteq N[y]$; equivalently, G is a QT-graph if and only if G has no induced subgraph isomorphic to P_4 or C_4 [15, 23, 30, 31]. The class of QT-graphs is a subclass of the class of cographs [10, 11] and contains the class of threshold graphs [9].

Many researchers have devoted their work to the study of QT-graphs. Wolk [30] called these graphs *comparability graphs of trees* and gave characterization of them. Golumbic [15] called them *trivially perfect* graphs with respect to a concept of "perfection". Ma, Wallis and Wu [23] called them *quasi-threshold* graphs (QT-graphs) and studied algorithmic properties.

The class of QT-graphs is a subclass of the well-known class of perfect graphs [6, 16, 24]; 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. [23] presented polynomial algorithms for a number of optimization problems. In particular, they gave an $O(nm)$ time algorithm for the recognition problem, and polynomial algorithms for the Hamiltonian cycle problem and the bandwidth problem. They also gave a formula for the clique covering number and conditions for a QT-graph to be Hamiltonian. Yan et. al. [33] stated important characterizations of these graphs and presented a linear-time algorithm, that is, $O(n + m)$, for the recognition problem. They also gave linear-time algorithms for the edge domination problem and the bandwidth problem in this class of graphs.

To the best of our knowledge, no parallel algorithms for Hamiltonian problems on QT-graphs are available in the literature. On the other hand, a variety of parallel algorithms have been described for optimization and combinatorial problems on many other classes of perfect graphs; for parallel algorithms on cographs and threshold graphs, see [1, 3, 12, 19, 22, 26].

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. We also 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 construct algorithms for finding a Hamiltonian cycle and computing the Hamiltonian completion number and a Hamiltonian completion edge set of a QT-graph; our algorithms take $O(\log^2 n)$ time and require $O(n + m)$ processors on the CREW PRAM model. In addition, we show 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. This implies an $O(\log n)$-time parallel algorithm for recognizing whether a QT-graph is a Hamiltonian graph. 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 [4, 20, 28], 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 $O(\log^2 n)$ time with $O(n + m)$ processors.

(iii) The Hamiltonian completion number of a QT-graph can be computed in $O(\log n)$ time with $O(n + m)$ processors.

(iv) Hamiltonian QT-graphs can be recognized 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; that is, the maximum clique problem, the maximum independent set problem, the clique cover problem and the coloring problem.

We should point out that, to the best of my knowledge, the study of the Hamiltonian completion number on QT-graphs has not received much attention. On the other hand, this problem on other classes of graphs has been extensively studied (see [2, 13, 21, 27]).

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 and 5 we present parallel algorithms for constructing tree representations of a QT-graph. Based on these representations and the conditions of Section 3, we present the main results of the paper in Sections 6 and 7; we design and analyze parallel algorithms for finding a Hamiltonian cycle and computing the Hamiltonian completion number and the Hamiltonian completion edge set of a QT-graph. In Section 8 we show that other optimization problems on QT-graph can be efficiently solved in parallel. Finally, in Section 9 we conclude with a summary of our results and extensions.

2. Quasi-threshold Graphs and their Structures

Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. The neighbourhood of a vertex x is the set $N(x) = N_G(x)$ consisting of all the vertices of G which are adjacent with x. The closed neighbourhood of x is defined by $N[x] = N_G[x] := \{x\} \cup N(x)$. The subgraph of a graph G induced by a subset $S \subseteq V(G)$ is denoted by $G[S]$. Let X and Y be two subsets of a certain set. Then $X \subset Y$ means that X is a proper subset of Y, and if $Y \subseteq X$, then let $X - Y$ denote $X \setminus Y$.

For a vertex subset S of a graph G, we define $G - S$ by $G[V(G) - S]$. 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 ([26]). If G is a QT-graph, then for every subset $S \subset V(G)$, both $G[S]$ and $G - S$ 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 ([26]). The following three statements hold.

(i) A graph 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) A graph G is a QT-graph if and only if $G - cent(G)$ is a QT-graph.

(iii) Let G be a connected QT-graph. If $G - cent(G) \neq \emptyset$, then $G - cent(G)$ contains at least two connected components.

Let G be a connected QT-graph. Then $V_1 := cent(G)$ is not an empty set by Theorem 1. Put $G_1 := G$, and $G - V_1 = G_2 \cup G_3 \cup \ldots \cup G_r$, where each G_i is a connected component of $G - V_1$ and $r \geq 3$. Then since each

G_i is an induced subgraph of G, G_i is also a QT-graph, and so let $V_i := cent(G_i) \neq \emptyset$ for $2 \leq i \leq r$. Since each connected component of 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 $V(G)$.

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

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

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

It is easy to see that the above partition of $V(G)$ possesses the following properties.

Theorem 2.2 ([26]). Let G be a connected QT-graph, and let $V(G) = V_1 + V_2 + \ldots + V_k$ be the partition defined above; in particular, $V_1 := cent(G)$. Then this partition and the partially ordered set $(\{V_i\}, \leq)$ have the following properties:

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

(P2) For every V_j, $cent(G[\{\cup V_i \mid V_i \leq V_j\}]) = V_j$.

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

(P4) Every edge with both endpoints in V_i is a free edge.

(P5) 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.

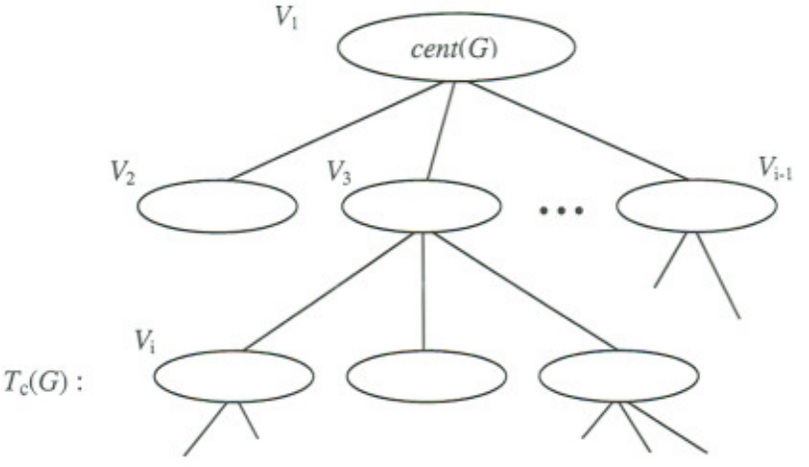

Figure 1. The typical structure of the *cent-tree* $T_c(G)$ of a QT-graph.

The results of Theorem 1.2 provide algorithmic and structural properties for the class of QT-graphs. A typical structure of such a QT-graph G is shown in Figure 1. We shall refer to the structure that meets the properties of Theorem 1.2 as *cent-tree* of the graph G and denote it by $T_c(G)$. The cent-tree is a rooted tree with root V_1; every node V_i of the tree $T_c(G)$ is either a leaf or has at least two children. Moreover, $V_s \leq V_t$ if and only if V_s is an ancestor of V_t. Thus, we can state the following result.

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

If V_i and V_j are disjoint vertex sets of a QT-graph G, we say that V_i and V_j are *clique-adjacent* and denote $V_i \approx V_j$ if $V_i \le V_j$ or $V_j \le V_i$.

Let G be a QT-graph and let $V = V_1 + V_2 + \ldots + V_k$ 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 $(\{V_i\}, \le)$ or, equivalently, V_t is a leaf node of $T_c(G)$, $s \le t \le q$. 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 $\{\cup V_i \mid V_1 \le V_i \le V_t\}$, for every maximal element V_t of $(\{V_i\}, \le)$, provide a clique cover of size $\kappa(G)$ which has the property to be a smallest possible clique cover of G; that is $\alpha(G) = \kappa(G)$. Based on the Theorem 2 or, equivalently, on the cent-tree of G, it is easy to show that the clique number $\omega(G)$ equals the chromatic number $\chi(G)$ of G; that is, $\chi(G) = \omega(G)$.

3. Hamiltonian *QT*-graphs

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

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

where p is the number of children of the node V_i $(1 \le i \le k)$. Figure 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-label*$(V_i) = 2$ if V_i is an internal node or *H-label*$(V_i) = 1$ if V_i is the root of the tree, *H-label*$(V_{i1}) = 1$, *H-label*$(V_{i2}) = -1$, *H-label*$(V_{i3}) = 0$, and *H-label*$(V_{i4}) = 0$. We shall show that G is a Hamiltonian QT-graph if *H-label*$(V_i) \ge 0$ for each node $V_i \in T_c(G)$.

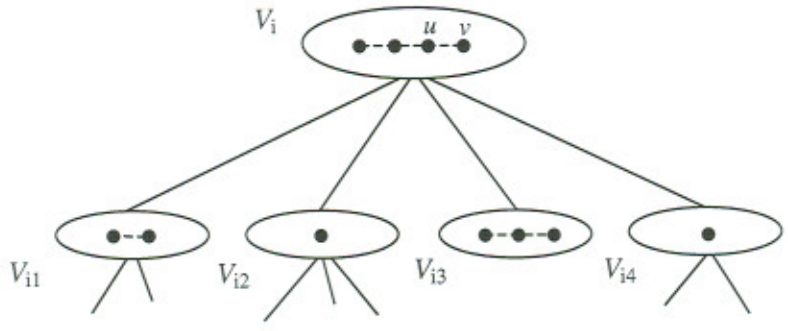

Figure 2. A node of the *cent-tree* $T_c(G)$ of a QT-graph along with its four children; the vertices of each node of $T_c(G)$ are denoted by black bullets.

Let $V_{i1}, V_{i2}, \ldots, V_{ip}$ be the children of an internal node V_i of the cent-tree $T_c(G)$ such that *H-label*$(V_i) \ge 0$, and let *list*$(V_i) = (v_{i1}, \ldots, v_{i(p-1)}, v_{ip}, \ldots, v_{is})$ be the list of the vertices of the node V_i, where $p \ge 2$ and

$s \geq p$-1. Let *a-vertices*(V_i) = (v_{ip}, $v_{i(p+1)}$, ..., v_{is}); the elements of this list *a-vertices*(V_i) are called *available vertices* of the node V_i. If V_i is the root of the cent-tree then *a-vertices*(V_i) = ($v_{i(p+1)}$, $v_{i(p+2)}$, ..., v_{is}). In Figure 2, for the internal node V_i we have *a-vertices*(V_i) = {u, v}.

Let $V_{f(1)}$, $V_{f(2)}$, ..., $V_{f(t)}$ 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 $2 \leq f(i) \leq k$ and $1 \leq i \leq t$-1. We define the *h-sequence* of the cent-tree $T_c(G)$ to be the following sequence:

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

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 G is $2t$.

By definition there exists no pair $V_{f(i)}$, $V_{f(j)}$ of elements $V_{f(1)}$, $V_{f(2)}$, ..., $V_{f(t)}$ of the h-sequence($T_c(G)$) such that $V_{f(i)} = V_{f(j)}$ for $i \neq j$, $1 \leq i, j \leq t$. On the other hand, may exist elements $V_{a(i1)}$, $V_{a(i2)}$, ..., $V_{a(iq)}$ such that $V_{a(i1)} = V_{a(i2)} = \ldots = V_{a(iq)} = V_i$, where V_i is an internal node $T_c(G)$; q is equal to the number of children of V_i minus 1. Let $a(i1)$ and $a(iq)$ be the indices of the leftmost and rightmost occurrence of V_i in h-sequence($T_c(G)$), and let $a(i1) < a(i2) < \ldots < a(iq)$. We say that $V_{a(i1)}$ is the first occurrence of V_i, $V_{a(i2)}$ is the second occurrence of V_i, and so on; $V_{a(iq)}$ is the last occurrence of V_i in h-sequence($T_c(G)$). 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 have the following result.

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

Let G be a *QT*-graph and h-sequence($T_c(G)$) be its h-sequence. We next use a depth-first search (dfs) traversal strategy for searching the graph G and building a spanning tree of G. 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; it works as follows:

Traversal strategy h-dfs:

(i) Compute the h-sequence ($V_{f(1)}$, $V_{a(1)}$, $V_{f(2)}$, ..., $V_{f(t)}$, $V_{a(t)} = V_1$) 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) If v is a visited vertex and $v \in V_{f(i)}$, then visit in turn each unvisited vertex of $V_{f(i)}$ ($1 \leq i \leq t$);

(iv) Once all the vertices of $V_{f(i)}$ have been visited, select an unvisited vertex v_i from the node $V_{a(i)} = V_i$ of the h-sequence, $1 \leq i \leq t$; visit v_i, mark v_i visited and if $V_{a(i)}$ is the last occurrence of V_i in h-sequence, then visit in turn each unvisited vertex of $V_{a(i)}$; otherwise, does to $V_{f(i+1)}$ and select an unvisited vertex from this set and visit it.

(v) Continue until the last vertex u of the rood node V_1 becomes a visited vertex;

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

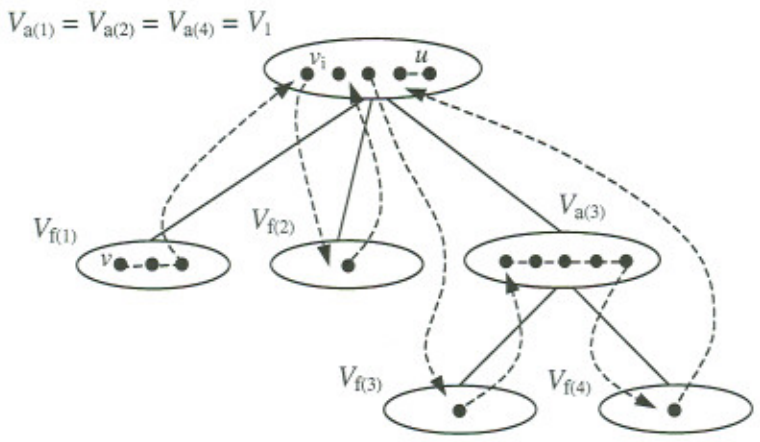

Figure 3. The h-dfs traversal of a *cent-tree* $T_c(G)$ of a QT-graph G.

It is well-known that if G is a connected undirected graph, then the dfs forest of G contains only one tree. Moreover, it is obvious that if each node of the dfs tree 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); G contains a Hamiltonian cycle if the root of the dfs tree and the unique leaf are adjacent in G. We next prove the following result.

Lemma 3.1. A QT-graph G is a Hamiltonian graph if $H\text{-}label(V_i) \geq 0$ for each node $V_i \in T_c(G)$.

Proof. Let V_1, V_2, ..., V_k be the nodes of the cent-tree $T_c(G)$ of the QT-graph G rooted at V_1, and let ($V_{f(1)}$, $V_{a(1)}$, $V_{f(2)}$, ..., $V_{f(t)}$, V_1) be the h-sequence of $T_c(G)$. Let $q(i)$ be the number of all the nodes of h-sequence(T_c), say, $V_{a(i1)}$, $V_{a(i2)}$, ..., $V_{a(iq)}$, such that $V_{a(i1)} = V_{a(i2)} = ... = V_{a(iq)} = V_i$, where V_i is an internal node $T_c(G)$; that is, V_i is the *lca* of some pairs of leaves of $T_c(G)$. By definition, $q(i)$ is equal to the number of children of V_i minus 1.

Let V_{i1}, V_{i2}, ..., V_{ip} be the children of the node V_i and let $list(V_i) = (v_{i1}, ..., v_{i(p-1)}, v_{ip}, ..., v_{is})$. Then, $q(i) = p - 1$. Since $H\text{-}label(V_i) \geq 0$, it follows that the V_i contains at least p-1 vertices; note that, it contains p vertices if V_i is the root V_1 of the cent-tree $T_c(G)$.

We select a vertex v from $V_{f(1)}$ and we perform an h-dfs traversal to G starting at v. Since each node V_i contains at least p - 1 vertices (p vertices if $V_i = V_1$) and $q(i) = p$ - 1 ($q(i) = p$ if $V_i = V_1$ because the last element of the h-sequence is the root V_1 of the cent-tree), 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 $V_{a(i)}$, $1 \leq i \leq t$-1; this is also true for the nodes $V_{f(t)}$ and V_1. On the other hand, the nodes $V_{a(i)}$ and $V_{f(i+1)}$ are clique-adjacent. Thus, the h-dfs tree of G has the property that each node has at most one child; that is, G contains a Hamiltonian path. Moreover, $V_{f(1)}$ 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 $V_i \leq V_j$ and $H\text{-}label(V_i) > 0$ and $H\text{-}label(V_j) < 0$. Let u be an available vertex of the node V_i. We define an operation that moves the available vertex u from the node V_i to node V_j. We call this operation *vertex-move*, or *v-move* for short.

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 $V_s \leq V_t$, $G[\{\cup V_i \mid V_s \leq V_i \leq V_t\}]$ is a complete graph. Obviously, if V_t is a maximal element of $(\{V_i\}, \leq)$, then after applying a *v*-move operation the graph $G[\{\cup V_i \mid V_1 \leq V_i \leq V_t\}]$ 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 $T_c(G)$ be the cent-tree of G. The graph G is a Hamiltonian QT-graph if and only if either $H\text{-}label(V_i) \geq 0$ for each node $V_i \in T_c(G)$ or we can construct an h-tree $T_h(G)$ such that $H\text{-}label(V_i) \geq 0$ for each node $V_i \in T_h(G)$.

Proof. The if implication follows directly from Lemma 3.1 since $H\text{-}label(V_i) \geq 0$ for each node $V_i \in T_h(G)$. Note that Proposition 3.1 also holds for the h-tree $T_h(G)$.

Suppose now that there exist nodes in $T_h(G)$ with negative H-labels. Let V_i be such a node and let each ancestor V_{ij} of V_i has $H\text{-}label(V_{ij}) \geq 0$; note that the leaves of the tree $T_h(G)$ have zero H-labels. Since $H\text{-}label(V_i) < 0$, it follows that there exists no predecessor V_{ik} of V_i with available vertices; that is, $H\text{-}label(V_{ik}) \leq 0$.

Let $V_{i1}, V_{i2}, ..., V_{ip}$ be the children of V_i and let q be the number of vertices $v_{i1}, v_{i2}, \ldots, v_{iq}$ of V_i, where $q < p - 1$ if V_i is an internal node and $q < p$ if V_i is the root of the tree $T_h(G)$. We construct the h-dfs tree of G rooted at vertex v; recall that the starting vertex v belongs to $V_{f(1)}$. We consider the following two cases: (i) $q < p - 1$. It is easy to see that the vertex v_{iq} has at least two children in the h-dfs tree. (ii) $q = p - 1$. In this case V_i is the root of the tree $T_h(G)$; that is, $V_i = V_1$, and each vertex in the h-dfs tree has only one child except, of course, of the unique leaf u. The vertices v and u are not adjacent in G since they do not belong to the same connected comport of the graph $G - V_1$. Thus, in both cases the graph G does not contain a Hamiltonian cycle. □

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.

Let G be a QT-graph and let $T_c(G)$ be its cent-tree with node set $\{V_1, V_2, ..., V_k\}$ and root V_1. We have shown that if node V_i is an ancestor of node V_j in the cent-tree of G, then V_i and V_j are clique-adjacent. Thus, if $(V_1, V_2, \ldots, V_i)$ is a path from the root V_1 of the cent-tree to a node V_i, then $deg(V_1) > deg(V_2) > \ldots > deg(V_i)$, where $deg(V_i)$ 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 $deg(v_1) \geq deg(v_2) \geq \ldots \geq deg(v_p)$, $1 \leq p \leq n$; see also [33].

Based on this property, we describe a method that produces a tree representation of a QT-graph; see also [33]. We call this tree *degree-tree* of G, or d-tree for short, and we denote it by $T_d(G)$. The method is as follows. First, sort the vertices $v_1, v_2, ..., v_n$ of G according to their degrees; let $D = (v_1, v_2, ..., v_n)$ be a sequence such that $deg(v_1) \geq deg(v_2) \geq ... \geq deg(v_n)$. Then, construct the tree T_g with vertex set $\{v_1, v_2, ..., v_n\}$ in the following manner: for every vertex $v_i \in D$, $2 \leq i \leq n$, find the vertex v_k, if it exists, such that k is the maximum index satisfying $1 \leq k < i$ and (v_k, v_i) is an edge in G; add the edge (v_k, v_i) into $E(T_g)$. Finally, root the tree T_g at vertex $r = v_1$. The resulting tree is the d-tree $T_d(G)$ of the QT-graph G.

We next describe the above algorithm in a more formal and parallel way; it takes as input a QT-graph G and produces the d-tree $T_d(G)$.

Algorithm Degree-Tree-Construction (DT_CON):

Step 1. **Compute** the degree $deg(v_i)$ for each vertex $v_i \in V$;

Step 2. **Sort** the vertices $v_1, v_2, ..., v_n$ of G according to their degrees;
Let $D = (v_1, v_2, ..., v_n)$ be a sequence such that $deg(v_1) \geq deg(v_2) \geq ... \geq deg(v_n)$;

Step 3. **Construct** the tree T_g with vertex set $\{v_1, v_2, ..., v_n\}$ as follows:

3.1 Set $V(T_g) \leftarrow \{r = v_1, v_2, ..., v_n\}$;

3.2 For every vertex $v_i \in D$, $2 \leq i \leq n$, do in parallel

3.2.1 find the vertex v_k, if it exists, such that: k is the maximum index satisfying $1 \leq k < i$ and (v_k, v_i) is an edge in G;

3.2.2 add the edge (v_k, v_i) into $E(T_g)$;

Step 4. **Root** the tree T_g at vertex $r = v_1$; the rooted tree T_g is the degree-tree $T_d(G)$ of G;

end.

Let us now compute the time-processor complexity of the proposed parallel algorithm for constructing the degree-tree of a QT-graph. We shall use a step-by-step analysis.

Step 1: Let $\langle u_1, u_2, ..., u_{d_i} \rangle$ be the list of vertices adjacent to v_i, where d_i is the degree of v_i. Then, d_i can be computed in $O(\log d_i)$ time with $O(d_i)$ processors on the EREW PRAM model using standard path doubling techniques on linked lists. Since $\Sigma_{v_i \in V} d_i = O(m)$, this step is executed in $O(\log n)$ time using a total of $O(n + m)$ processors.

Step 2: It is well-known that n elements can be sorted in $O(\log n)$ time with $O(n)$ processors on the EREW PRAM model [4, 20].

Step 3: Let $D = (v_1, v_2, ..., v_n)$ be the sequence computed in Step 2, and let $N(v_i) = \{u_1, u_2, ..., u_{d_i}\}$ be the set of vertices adjacent to v_i $(1 \leq i \leq n)$. For each vertex v_i we compute the vertex u, if it exists, having the following property: $u \in N(v_i)$ 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 [4]) in $O(\log n)$ time with $O(n)$ processors on the CREW PRAM. Thus, the whole step is executed in $O(\log n)$ time using a total of $O(n)$ processors.

Step 4: The problem of rooting the tree T_g at the vertex $r = v_1$ (for each vertex $v \neq r$, we determine the parent $p(v)$ of v when T_g is rooted at r) 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 [4, 20, 28]. Thus, this step can be performed within the stated bounds.

Taking into consideration the time and processor complexity of each step of the algorithm we have the following result.

Lemma 3.1. 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.

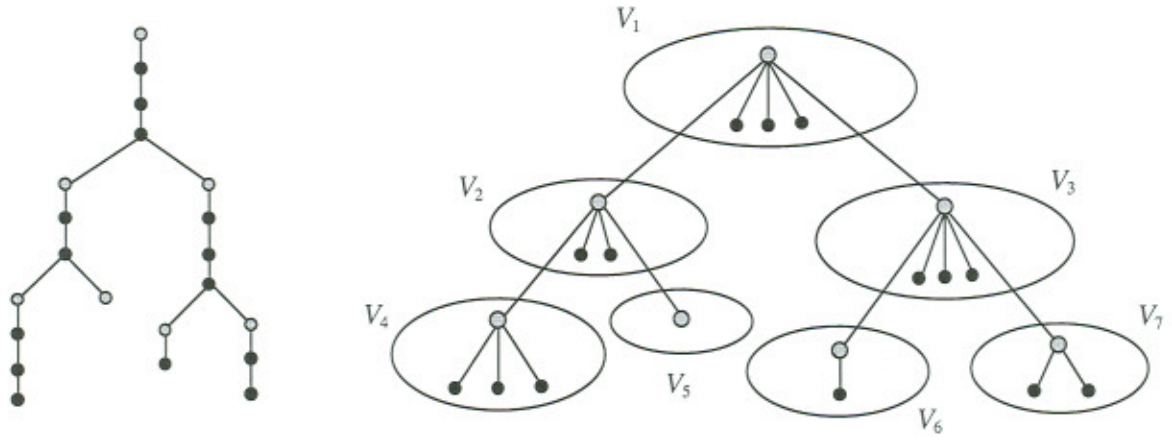

Figure 4. The degree-tree $T_d(G)$ and the *cent-tree* $T_c(G)$ of a QT-graph G.

Based on the structural properties of the d-tree $T_d(G)$ of a QT-graph, 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 set V_i of the cent-tree of G if and only if u is a unique child of the vertex $p(u)$ in the d-tree $T_d(G)$; see Figure 4. Let u_2, ..., u_k be the vertices of the d-tree $T_d(G)$ with the property that their parents have at least two children and let $R = \{r = u_1\ u_2, ..., u_k\}$, where r is the root of the d-tree. It is easy to see that, the cent-tree $T_c(G)$ has nodes V_1, V_2, ..., V_k and $u_i \in V_i$, $1 \le i \le k$. 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 $T_d(G)$ that belongs to R. The vertex $u \notin R$ of the graph G belongs to the node set V_i if the least ancestor of u in $T_d(G)$ that belongs to R is the vertex u_i, $1 \le i \le k$; see Figure 4.

More precisely, we have the following parallel algorithm; it takes as input a QT-graph G and produces the cent-tree of the graph G.

Algorithm Cent-Tree-Construction (CT_CON):

Step 1. **Compute** the d-tree $T_d(G)$ with vertex set $\{r = v_1, v_2, ..., v_n\}$ using Algorithm QT_CON;

Step 2. **For** each vertex $v_i \in T_d(G)$, $1 \le i \le n$, 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 $r = u_1, u_2, ..., u_k$ be the red vertices of T_g, $k \ge 1$;

Step 3. **For** each vertex $v_i \in T_d(G)$, $2 \le i \le n$, do in parallel
Find the least ancestor u_j of vertex v_i with read color and set $p(v_i) \leftarrow u_j$;
Let T_g be the resulting tree; $r = u_1$ is the root of T_g; we set $p(r) \leftarrow r$;

Step 4. **For** each red vertex u_i construct a node set V_i and set $V_i \leftarrow \{u_i\}$, $1 \le i \le k$;

Step 5. **Construct** the tree graph T_c as follows:

5.1 Set $V(T_c) \leftarrow \{r_c = V_1, V_2, ..., V_k\}$;

5.2 For each red vertex $u_i \in T_g$, $2 \le i \le k$, do in parallel

if u_j is the parent of u_i, then add the edge (V_i, V_j) into $E(T_g)$;

Step 6. **Compute** the vertices of each node $V_1, V_2, ..., V_k$ of the tree T_c as follows:

For each black vertex $v_i \in T'_g$, $2 \le i \le k$, do in parallel

if u_j is the parent of v_i, then add the vertex v_i into node set V_j;

Step 7. **Root** the tree T_c at node V_1; the rooted tree T_c is the cent-tree $T_c(G)$ of the graph G;

end.

We next compute the time-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 the d-tree $T_d(G)$ of a graph on n vertices and m edges is constructed in $O(\log n)$ time with $O(n + m)$ processors on the CREW PRAM model using the recognition algorithm QT_CON.

Step 2: Obviously, the parent u of a vertex $v_i \in T_d(G)$, $2 \le i \le n$, has more than one child if there exist a vertex v_j such that $p(v_i) = p(v_j) = u$. Let $p(v'_1), p(v'_2), \ldots, p(v'_n)$ be the sorted sequence of the parents of the vertices of the tree $T_d(G)$; assume that $p(v_1) = v_1$. Then, the vertex v'_i has more than one child if $p(v'_i) = p(v'_{i-1})$ or $p(v'_i) = p(v'_{i+1})$. Since n elements can be sorted in $O(\log n)$ time with $O(n)$ processors on the EREW PRAM model, this step can be executed within the same time and processor bounds.

Step 3: The tree T_g can be computed by using the pointer jumping technique on T_g; for each vertex v_i of T_g $(1 \le i \le n)$ such that $p(v_i)$ is a black vertex, do the following: $p(v_i) \leftarrow p(p(v_i))$; continue, until $p(v_i)$ is a red vertex, for every $v_i \in T_g$ (this is the well-known parallel prefix algorithm [4, 20]). Thus, T_g can be computed in $O(\log n)$ time with $O(n)$ processors on the CREW PRAM model.

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

Step 6: It is easy to see that the elements of the node sets $V_1, V_2, ..., V_k$ can be computed from the tree T_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. Obviously, this computation can be executed in $O(1)$ parallel time with $O(n)$ processors the EREW PRAM model.

Step 7: We apply the Euler tour technique on the tree graph T_c to solve the problem of rooting T_c at the vertex V_1; that is, for each vertex $V_i \neq V_1$, we determine the parent $p(V_i)$ of V_i when T_c is rooted at V_1. As we have seen, we can root the tree T_c in $O(\log n)$ time with $O(n / \log n)$ processors on the EREW PRAM.

Therefore, from the previous step-by-step analysis, it follows that the construction algorithm CT_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:

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. Construction of an *H*-tree of a Hamiltonian *QT*-graph

Let G be a QT-graph and let $T_c(G)$ be its cent-tree with nodes V_1, V_2, ..., V_k rooted at V_1. We have proved that if G is a Hamiltonian graph then either $H\text{-}label(V_i) \geq 0$ for each node $V_i \in T_c(G)$ or we can construct an h-tree $T_h(G)$ such that $H\text{-}label(V_i) \geq 0$ for each node $V_i \in T_h(G)$; see Theorem 3.1.

Consider the case where the cent-tree of a Hamiltonian QT-graph G has nodes, say, V'_1, V'_2, ..., V'_p ($p < k$) with H-labels less than zero. In this case, we are interested in constructing an h-tree $T_h(G)$ of the cent-tree $T_c(G)$. This can be done by moving available vertices from certain nodes of $T_c(G)$ to nodes V'_1, V'_2, ..., V'_p using the v-move operation (see Section 3). Recall that, if a Hamiltonian graph G has a node, say, V_i, such that $H\text{-}label(V_i) < 0$, then there exists an ancestor V_j of V_i in the cent-tree $T_c(G)$ such that $H\text{-}label(V_j) > 0$; that is, a node V_j which 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 they contain. Next we determine the nodes of the cent-tree $T_c(G)$, say, V'_1, V'_2, ..., V'_p ($p < k$), with H-labels less than zero and compute the number of available vertices we have to v-move to each of these nodes so that $H\text{-labels}(V'_1) = H\text{-labels}(V'_2) = \ldots = H\text{-labels}(V'_p) = 0$; let n_1, n_2, ..., n_p be these numbers. We add n_i dummy vertices to node V'_i ($1 \leq i \leq p$), and we assign to each of these dummy vertices an available vertex; (see Figure 5(a); the available and the dummy vertices are denoted with grey and white bullets, respectively). Hereafter, an available vertex which has been assigned to a dummy vertex will be referred to as an *m-vertex*. Note that, there is a one-to-one correspondence between the m-vertices and the dummy vertices. Finally, we move the m-vertices to appropriate nodes of the cent-tree and delete the dummy vertices from the tree.

It is easy to see that the crucial step of the above construction method is the step in which we assign available vertices to dummy vertices. This step is efficiently implemented as follows:

(i) We construct the degree-tree $T_d(G)$ of the input graph G and paint the available vertices with grey color; initially, all the vertices of the tree are black. Then, we remove the black vertices (non-available) from the tree $T_d(G)$ using standard pointer jumping techniques. The resulting tree contains all the available vertices of the cent-tree $T_c(G)$; we call it *available-tree* of G.

(ii) Let dv_{i1}, dv_{i2}, ..., dv_{id} be the dummy vertices of a node V_i and let V_j be the least ancestor of V_i in the cent-tree such that $H\text{-}label(V_j) > 0$; let $a\text{-}vertices(V_j) = (v_{ip}, v_{i(p+1)}, \ldots, v_{is})$. Initially, we assign all the dummy vertices dv_{i1}, dv_{i2}, ..., dv_{id} to node v_{ij} of the list $a\text{-}vertices(V_j)$, which in the available-tree has the longest distance from the root of the tree. We implement this assignment by using the children-parent relation on the available-tree; more precisely, we make the dummy vertices dv_{i1}, dv_{i2}, ..., dv_{id} to be children of the vertex v_{ij} in the available-tree (see Figure 5(b); the available-tree of G along with its dummy vertices). We mark as m-vertices all the vertices of the available-tree that have children dummy vertices.

(iii) Then, we check if each of the m-vertices in the available-tree has exactly one dummy child. If so, a one-to-one correspondence between the m-vertices and the dummy vertices has been achieved.

Consider now the case where there exist m-vertices in the available-tree that have more than one dummy child. Let v_{ij} be such an m-vertex and let $(dv_{i1}, dv_{i2}, \ldots, dv_{id})$ be the list of its dummy children, where $d > 1$. In this case, we determine d-1 vertices in the available-tree such that they are ancestors of the m-vertex v_{ij} and have no dummy children; let u_{i2}, ..., u_{id} be these vertices. Then, we make the dummy vertex dv_{ij} to be a child of the vertex u_{ij}, $2 \leq j \leq d$.

We continue (iii) until we achieve a one-to-one correspondence between the m-vertices and the dummy vertices in the available-tree.

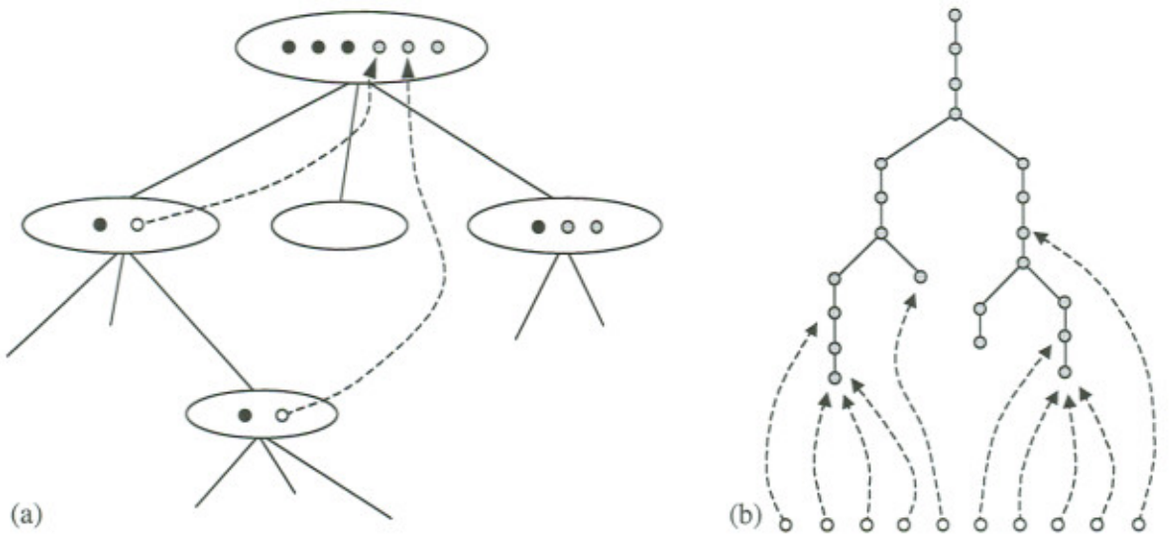

Figure 5. (a) Two dummy (white) vertices and their corresponding (grey) available vertices of the cent-tree of a QT-graph G; (b) The available-tree of G along with its dummy vertices.

We next present a parallel algorithm for the construction of the h-tree $T_h(G)$ of the cent-tree of G. Since G is a Hamiltonian QT-graph, the algorithm always succeeds in computing the h-tree $T_h(G)$.

Algorithm H-Tree-Construction (HT_CON):

Step 1. **Compute** the cent-tree $T_c(G)$ of a Hamiltonian QT-graph G using Algorithm CT_CON, and the H-labels of each node $V_i \in T_c(G)$, $1 \le i \le k$; note that $r_c = V_1$;

Step 2. **If** $H\text{-}label(V_i) \ge 0$ for each node $V_i \in T_c(G)$, $1 \le i \le k$, then
$T_c(G)$ is an h-tree; set $T_h(G) \leftarrow T_c(G)$ and exit;

Step 3. **Compute** the degree-graph $T_d(G)$ using Algorithm DT_CON;
Paint the root r of the tree grey and set $p(r) \leftarrow r$; initially, all the vertices of $T_d(G)$ are black;

Step 4. **For** each non-leaf node $V_i \in T_c(G)$, $1 \le i \le k$, do in parallel
Let $V_{i1}, ..., V_{ij}, ..., V_{ip}$ be the children of the node V_i, $p \ge 2$, and let $list(V_i) = (v_{i1}, ..., v_{i(p-1)}, v_{ip}, ..., v_{is})$;
4.1 If $H\text{-}label(V_i) > 0$ then
paint the vertex v_{ij} of $T_d(G)$ grey, if v_{ij} is an available vertex of the node set V_i, that is, $v_{ij} \in a\text{-}vertices(V_i)$;
4.2 If $H\text{-}label(V_i) < 0$ then
construct the set $D_i = \{dv_{i1}, dv_{i2}, ..., dv_{id}\}$ of dummy vertices, where $d = p - s - 1$;
paint the dummy vertices $dv_{i1}, dv_{i2}, ..., dv_{id}$ white, and
add them in $T_d(G)$ by setting $p(dv_{ij}) \leftarrow v_{is}$, $1 \le j \le d$;
Let T'_g be the resulting tree;

Step 5. **For** each black vertex $u_i \in T'_g$, do in parallel

5.1 Find the least grey ancestor w_i of v_i, set $p(u_i) \leftarrow w_i$, and delete the black vertex u_i from T'_g;

Let T_g be the resulting tree; it contains only grey and white vertices; that is, T_g is the available-tree of G with its dummy vertices; note that, its root r is a grey vertex but it is not an available vertex.

Step 6. **While** there exist dummy vertices having the same (grey) parent, execute Steps 7 through 9;

Step 7. **Compute** the level function on the tree T_g rooted at vertex r;

Step 8. **For** each grey vertex $v_i \in T_g$, do in parallel

8.1 Construct the list $(dv_{i1}, dv_{i2}, ..., dv_{iq})$ of its dummy children;

8.2 Rank the list $(dv_{i1}, dv_{i2}, ..., dv_{iq})$;

8.3 If $q \geq 2$ then find the ancestor u_{ij} of v_i at distance j-1 and set $p(dv_{ij}) \leftarrow u_{ij}$ for $2 \leq j \leq q$; mark u_{ij} as m-vertex;

8.4 If $q \geq 1$ then mark v_i as m-vertex;

Step 9. **For** each vertex $u_i \in T_g$ such that u_i is an m-vertex or $p(u_i)$ is an m-vertex, do in parallel

9.1 Find the least grey ancestor w_i of u_i and set $p(u_i) \leftarrow w_i$;

9.2 Compute the set $\{du_{i1}, du_{i2}, ..., du_{iq}\}$ of the dummy children of vertex u_i; note that, if u_i is not an m-vertex (that is, $p(u_i)$ is an m-vertex), then this set is empty;

9.3 If $q \geq 2$ then set $p(du_{ij}) \leftarrow w_i$ for $2 \leq j \leq q$;

Step 10. **Construct** the h-tree $T_h(G)$ from the cent-tree $T_c(G)$ as follows:

For each set $V_i \in T_c(G)$ such that $H\text{-}label(V_i) < 0$, do in parallel

set $V_i \leftarrow V_i \cup \{p(dv_{i1}), p(dv_{i2}), ..., p(dv_{id})\}$, where $dv_{ij} \in D_i$, $1 \leq j \leq d$;

delete the vertex $p(dv_{ij})$ from V_t $(1 \leq t \leq k)$, where V_t is an ancestor of V_i in $T_c(G)$;

end.

In order to determine the time and the number of processors required for the execution of the proposed parallel algorithm, we shall use a step-by-step analysis.

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 Algorithms CT_CON.

The number of children $V_{i1}, V_{i2}, ..., V_{ip}$ of a node $V_i \in T_c(G)$, $1 \leq i \leq k$, can be computed in $O(\log p)$ time with $O(p / \log p)$ processors on the EREW PRAM. Thus, it is easy to see that the H-label of V_i is computed in $O(\log s)$ time with $O(s / \log s)$ processors on the EREW PRAM, where s is the number of vertices in V_i. It follows that the H-labels of the cent-tree $T_c(G)$ are computed in $O(\log n)$ time with $O(n)$ processors on the EREW PRAM model.

Steps 2: Obviously, this step is executed in $O(\log n)$ time with $O(n / \log n)$ processors on the EREW PRAM model.

Step 3: The degree-tree $T_d(G)$ can be constructed in $O(\log n)$ time using $O(n + m)$ processors on the CREW PRAM model; see Algorithm DT_CON.

Step 4: Here, for each node $V_i \in T_c(G)$, $1 \leq i \leq k$, either Substep 4.1 or Substep 4.2 is executed. Substep 4.1. We have computed the number p of the children of each node V_i (see the analysis of Step 1). Then it is easy to see that we can find the available vertices of the node V_i by ranking the $list(V_i)$ of length s. It is well-known that the pointer jumping technique can be used to derive an EREW parallel algorithm

for the list-ranking problem; the corresponding running time is $O(\log s)$ and the corresponding number of processors is $O(s)$, where s is the number of vertices in V_i [4, 20, 28]. Thus, this substep is executed in $O(\log n)$ time with $O(n)$ processors on the EREW PRAM model. Substep 4.2. Obviously, this substep can be executed in $O(1)$ with $O(n)$ processors on the EREW PRAM model.

Step 5: The tree T_g can be computed by using the standard parallel pointer jumping technique on T'_g. Thus, T_g can be computed in $O(\log n)$ time with $O(n)$ processors on the CREW PRAM model.

Step 6: We can determine whether there exist two (at least) dummy vertices having the same parent $v_i \in T_g$ by computing the number q of the dummy children of v_i; we have seen that this computation needs $O(\log n)$ parallel time and $O(n)$ processors on the EREW PRAM model.

Step 7: It is well-known that finding the level of each vertex of a rooted tree with n vertices is a computation which can be done optimally in $O(\log n)$ time on the EREW PRAM model [4, 20, 28]. Thus, this step is executed in $O(\log n)$ time with $O(n / \log n)$ processors on the EREW PRAM model.

Step 8: This step consists of four substeps which are executed for each read vertex $v_i \in T'_g$. Substep 8.1. Let $dv_{i1}, dv_{i2}, \ldots, dv_{iq}$ be the children of the vertex v_i which are dummy vertices. Thus, the list $(dv_{i1}, dv_{i2}, \ldots, dv_{iq})$ can be constructed in $O(\log q)$ with $O(q)$ processors on the EREW PRAM by sorting its dummy children in increasing order. Substep 8.2. The list ranking of a list of length q can be done in $O(\log q)$ with $O(q)$ processors on the EREW PRAM. Substep 8.3. The ancestor of each vertex of a tree T with n vertices can be determined in $O(\log n)$ with $O(n / \log n)$ processors on the EREW PRAM by using the Euler-tour technique (this computation can also be done using the pointer jumping technique on T; note that in this case the computation needs CREW model). The distance of a vertex from the root of a tree T can be determined by computing the level function on T. In order to avoid concurrent writes during the coloring of the ancestor u_{ij} of the grey vertex v_i, we implement this substep as follows: for each vertex $u_{ij} \in T_g$ compute the number of its dummy children. If this number is positive then mark the vertex u_{ij} as m-vertex. Thus, this substep is executed in $O(\log n)$ with $O(n)$ processors on the EREW PRAM model. Substep 8.4. Obviously, this substep needs $O(1)$ time and $O(n)$ processors. Thus, we can easily conclude that the whole step is executed in $O(\log n)$ with $O(n)$ processors on the EREW PRAM model.

Step 9: Having computed the time-processor complexity of Step 8, it is easy to see that this step is executed in $O(\log n)$ time with $O(n)$ processors on the EREW PRAM model.

Step 10: The union of the vertex sets V_i and $\{p(dv_{i1}), p(dv_{i2}), \ldots, p(dv_{id})\}$, for all V_i such that *H-label*$(V_i) < 0$, is executed in $O(1)$ time with $O(n)$ processors on the EREW PRAM. The operations of determining the node set V_t such that $p(dv_{ij}) \in V_t$ and deleting $p(dv_{ij})$ from V_t, $1 \le t \le k$, requires $O(1)$ time and $O(n)$ processors on the EREW PRAM model; this computation can be done in constant time since, during the construction of the node sets $V_1, V_2, \ldots, V_k$ of the cent-tree $T_c(G)$, we can assign a label, say, i, in the vertex $v \in T_g$ if v belongs to V_i, $1 \le i \le k$.

It is easy to see than Steps 7 through 9 are executed k times; the value of k is determined subsequently. We prove the following lemma.

Lemma 5.1. Given a Hamiltonian quasi-threshold graph G, Algorithm HT_CON constructs an h-tree of G after $O(\log n)$ iterations of Steps 7 through 9.

Proof. Let T_g be the tree after an execution of Steps 7 through 9 and let $v_1, v_2, \ldots, v_m$ be the read vertices of T_g each of which has at least two dummy children. Hereafter, we shall call these vertices *active* grey vertices. The algorithm terminates the execution of Steps 7 through 9 when, after an iteration of these

steps, the tree T_g remains with no active grey vertices.

We observe that after an execution of Steps 7 through 9 no vertex $v_1, v_2, \ldots, v_m$ remains active. A non-active grey vertex w_i becomes active vertex if Step 8.3 generates vertices u_i having at least two dummy children and the vertex w_i is the least read ancestor of u_i. If so, then Steps 9.1 makes the vertex u_i child of w_i and Step 9.3 moves all but one dummy child from u_i to w_i. It is easy to see that, the vertex w_i becomes an active grey vertex if there exist $p \geq 1$ children of w_i having a total of $m \geq p + 2$ dummy children. If this is the case, then there exists a child of w_i having at least two dummy children.

Let u_i be such a child of the vertex w_i. Since u_i has more than one dummy children, it follows that at least two of the vertices $v_1, v_2, \ldots, v_m$ are needed for the generation of the vertex u_i. Thus, an execution of Steps 7 thought 9 halves the number of active read vertices in the tree T_g.

The above proves that no active grey vertex exists in the tree T_g after $O(\log n)$ iterations of Steps 7 through 9. That is, each grey (available) vertex in T_g has at most one dummy child. Thus, the tree $T_c(G)$ that is constructed in Step 10 is an h-tree of the cent-tree $T_c(G)$, and, thus, the lemma holds. □

From the previous step-by-step analysis and Lemma 5.1, it follows that the h-tree construction algorithm HT_CON runs in $O(\log^2 n)$ time using a total of $O(n + m)$ processors on the CREW PRAM model of computation. Thus, we have proved the following:

Theorem 5.1. An h-tree of a Hamiltonian QT-graph can be constructed in $O(\log^2 n)$ time with $O(n + m)$ processors on the CREW PRAM model.

6. Finding a Hamiltonian Cycle in a *QT*-graph

In Section 3 we proved necessary and sufficient conditions for a QT-graph to be Hamiltonian (see Theorem 3.1). Based on these conditions, we can construct a parallel algorithm for finding a Hamiltonian cycle in a Hamiltonian QT-graph.

Let us first describe a sequential algorithm for the problem. Let G be Hamiltonian QT-graph and let $T_c(G)$ be its cent-tree with nodes $V_1, V_2, \ldots, V_k$ and root V_1. If $H\text{-}label(V_i) \geq 0$ for each node $V_i \in T_c(G)$, then $T_h(G) := T_c(G)$; otherwise, we construct an h-tree of $T_c(G)$. Consider the h-sequence $(V_{f(1)}, V_{a(1)}, \ldots, V_1)$ of the tree $T_h(G)$ and construct the h-dfs tree of the graph G using the h-dfs traversal strategy on the tree $T_h(G)$. We select an arbitrary vertex v from the set $V_{f(1)}$ as start point. Since $H\text{-}label(V_i) \geq 0$ 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 Figure 7. 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). Moreover, it is easy to see that the h-sequence of the graph G can be also efficiently constructed in a parallel environment; 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 [4, 20]. Therefore, it is becoming obvious that we need an efficient parallel algorithm for the h-dfs traversal strategy, that is, an 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 dfs traversal; various graph numberings, including dfs, where the numbering algorithm is restricted to a particular order of traversal of the edges of the graph is P-complete [28].

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 G; it works as follows:

(i) Let $V(G)$ be the vertex set of the input graph. We first construct a directed graph F with $V(F) = V(G)$ and $E(F) = \varnothing$;

(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 $list(V_i) = (v_{i1}, v_{i2}, \ldots, v_{is})$ of each node V_i of the h-tree; we add the edge $\langle v_{ij}, v_{i(j+1)} \rangle$ in $E(F)$, $1 \leq j \leq s-1$;

(iii) Let $V_{f(i)}$, $V_{a(i)}$, $V_{f(i+1)}$ be three consecutive node in the h-sequence. Let $V_{a(i)}$ be the jth occurrence of $V_{a(i)}$ in the h-sequence and let $list(V_{a(i)}) = (v_{i1}, \ldots, v_{ij}, \ldots, v_{is})$. We compute the last vertex v_e and the first vertex v_h of the lists $list(V_{f(i)})$ and $list(V_{f(i+1)})$, respectively. Then, we add the edges $\langle v_e, v_{ij} \rangle$ and $\langle v_{ij}, v_h \rangle$ in $E(F)$, and delete the edge $\langle v_{i(j-1)}, v_{ij} \rangle$ from $E(F)$, $1 \leq j \leq s$.

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 Figure 6). Thus, Steps (i)-(iii) of the above method do not guarantee the construction of the h-dfs tree of the graph G. Figure 6 depicts the results of Steps (i)-(iii).

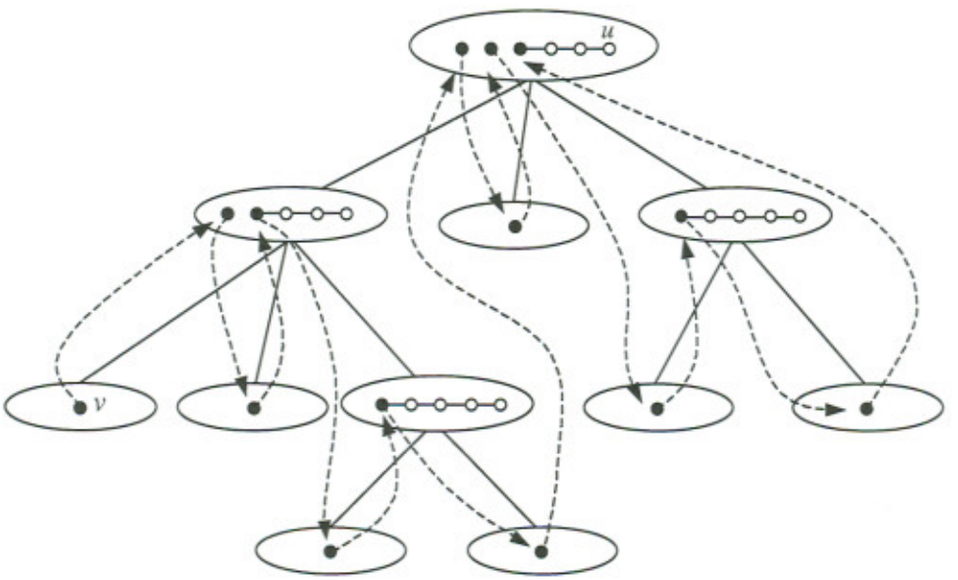

Figure 6. The results of Steps (i)-(iii) of the h-dfs tree construction of a QT-graph G.

Having computed the directed graph F using Steps (i)-(iii), let us now describe how we can modify the edge set of F so that the underline undirected graph of the resulting graph F produces the h-dfs tree of the graph G. We modify the edge set $E(F)$ as follows:

(iv) Let $V_{a(i)}$ be an internal node of the h-tree, $1 \leq i \leq t$. We first determine the vertex v_{ij} of its list $list(V_{a(i)}) = (v_{i1}, \ldots, v_{ij}, \ldots, v_{is})$, if it exists, with the property: there are vertices $x \in V_{a(i)}$ and $y \in V_{f(i+1)}$ such that $\langle v_{ij}, x \rangle$ and $\langle v_{ij}, y \rangle$ are edges in $E(F)$;

(v) Then, we delete the edge $\langle v_{ij}, y \rangle$ from $E(F)$, determine the last vertex v_{is} of $list(V_{a(i)})$ and add the edge $\langle v_{is}, y \rangle$ 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. Again, 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 is the h-dfs tree of the graph G (see Figure 7).

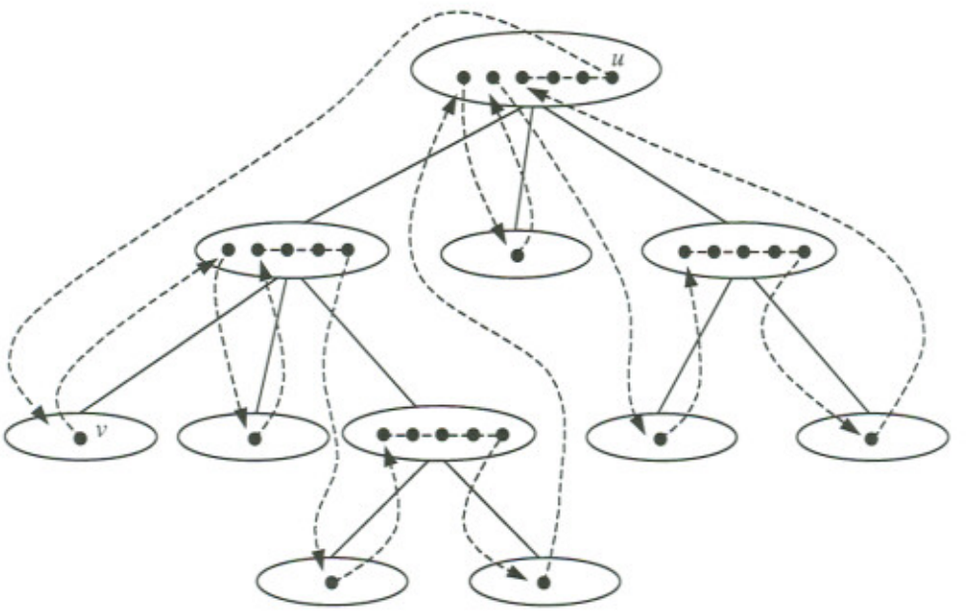

Figure 7. The structure of a Hamiltonian cycle of a QT-graph G; it is produced using the h-dfs traversal strategy on the h-tree $T_h(G)$.

Thus, we can produce a Hamiltonian path $(v = v_0, v_2, \ldots, v_n = u)$ of the 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, $(v, u) \in E(G)$. We add the edge $\langle v, u \rangle$ in $E(F)$ and take the underline undirected graph T of the resulting graph; T is a Hamiltonian cycle of G.

We next present in a more formal way the parallel algorithm for the construction of a Hamiltonian cycle of a QT-graph.

Algorithm Hamiltonian-Cycle-Construction (HC_CON):

Step 1. **Compute** the h-tree $T_h(G)$ of a Hamiltonian QT-graph G using Algorithm HT_CON;
Let $V_1, V_2, \ldots, V_k$ be the nodes of the tree T_h rooted at $r_h = V_1$;

Step 2. **Compute** the h-sequence of the h-tree $T_h(G)$; that is,
$h\text{-sequence}(T_h(G)) = (V_{f(1)}, V_{a(1)}, V_{f(2)}, \ldots, V_{f(t)}, V_{a(t)} = V_1)$,

Step 3. **Construct** a directed graph F with $V(F) = V(G)$ and $E(F) = \varnothing$,
and paint its vertices black;

Step 4. **For** each vertex $V_i \in T_h(G)$, $1 \le i \le k$, do in parallel
Construct the linked list $list(V_i) = (v_{i1}, v_{i2}, \ldots, v_{is})$ and
add the edge $\langle v_{ij}, v_{i(j+1)} \rangle$ in $E(F)$, $1 \le j \le s-1$;

Step 5. **For** each internal node $V_{a(i)} \in h$-sequence($T_h(G)$), $1 \le i \le t$, in parallel
If $V_{a(i)}$ is the jth occurrence of $V_{a(i)}$ in h-sequence($T_h(G)$), then
5.1 find the jth vertex v_{ij} of the $list(V_{a(i)}) = (v_{i1}, \ldots, v_{ij}, \ldots, v_{is})$;
5.2 add $\langle v_e, v_{ij}\rangle$ and $\langle v_{ij}, v_h\rangle$ in $E(F)$, where v_e and v_h are the last and the first vertices of $list(V_{f(i)})$ and $list(V_{f(i+1) \bmod t})$, respectively;
5.3 paint v_{ij} read;

Step 6. **For** each internal node $V_{a(i)} \in h$-sequence($T_h(G)$), $1 \le i \le t$, do in parallel
If v_{ij} and $v_{i(j+1)}$ are read vertices of $V_{a(i)}$, then delete edge $\langle v_{ij}, v_{i(j+1)}\rangle$ from $E(F)$;
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 $E(F)$ and add $\langle v_e, v_h\rangle$ in $E(F)$, where v_e and v_h are the last and the first vertices of $list(V_{a(i)})$ and $list(V_{f(i+1) \bmod t})$, respectively;

Step 7. **Construct** a spanning cycle C of G as follows:
Set $V(C) \leftarrow V(F)$;
For every edge $\langle v_i, v_j\rangle$ in $E(F)$, add (v_i, v_j) in $E(C)$;

end.

Let us now compute the time-processor complexity of the proposed parallel algorithm for the construction of the cent-tree of a QT-graph. We shall use a step-by-step analysis.

Step 1: The h-tree $T_h(G)$ of a QT-graph on n vertices and m edges is constructed in $O(\log^2 n)$ time with $O(n + m)$ processors on the CREW PRAM model using Algorithm HTC.

Step 2: Let h-sequence($T_h(G)$) = ($V_{f(1)}$, $V_{a(1)}$, $V_{f(2)}$, …, $V_{f(t)}$, $V_{a(t)}$). By definition, the node sequence $V_{f(1)}$, $V_{f(2)}$, ..., $V_{f(t)}$ is a left-to-right order listing of the leaves of the tree $T_h(G)$, and the node $V_{a(i)}$ is the lowest common ancestor of the nodes $V_{f(i)}$ and $V_{f(i+1)}$, where $2 \le f(i) \le k$ and $1 \le i \le t$-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 [4, 20]. We can also optimally compute the lowest common ancestor of two vertices of a rooted tree in $O(\log n)$ parallel time on the CREW PRAM model [4, 20]. 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; note that the cent-tree has k nodes, where $k \le n$.

Step 3: The directed graph F can be constructed in $O(1)$ time with $O(n)$ processors on the EREW PRAM model and its vertices can be painted within the same time-processor complexity.

Step 4: The list $list(V_i) = (v_{i1}, v_{i2}, \ldots, v_{is})$, $1 \le i \le k$, can be constructed in $O(\log s)$ time with $O(s)$ processors on the EREW PRAM model, since its elements can be sorted in $O(\log s)$ time with $O(s)$ processors on the same model of computation. Thus, the whole step can be executed in $O(\log n)$ time with $O(n)$ processors on the EREW PRAM model.

Step 5: Let $list(V_i) = (v_{i1}, \ldots, v_{ij}, \ldots, v_{is})$ be the list of the vertices of the node $V_i \in T_h(G)$, $1 \le i \le k$. 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 s vertices can be solved in $O(\log s)$ time with $O(s / \log s)$ processors on the EREW PRAM model. Thus, we can rank all the lists $list(V_i)$, $1 \le i \le k$, of the nodes of the tree $T_h(G)$ 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 6: It is easy to see that, this step is executed in $O(1)$ time with $O(n)$ processors on the CREW PRAM model.

Step 8: Obviously, this step is executed in $O(1)$ time with $O(n)$ processors on the EREW PRAM

model, since the connected directed graph F has $O(n)$ edges.

Therefore, from the previous step-by-step analysis, it follows that the Hamiltonian cycle construction Algorithm HC_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:

Lemma 6.1. A Hamiltonian cycle of a QT-graph can be constructed in $O(\log^2 n)$ time with $O(n + m)$ processors on the CREW PRAM model.

Corollary 6.1. If the cent-tree of a QT-graph G is an h-tree, then a Hamiltonian cycle of G can be constructed in $O(\log n)$ time with $O(n + m)$ processors on the CREW PRAM model.

7. Finding the Hamiltonian Completion Number of a QT-graph

It is well known that it is NP-complete to recognize whether a graph is Hamiltonian [5]. Based on the results of Section 3, we can construct parallel algorithms for computing the Hamiltonian completion number $hcn(G)$ and finding the Hamiltonian completion edge set $CE(G)$ of a QT-graph G. Obviously, if $hcn(G) = 0$ then G is a Hamiltonian graph. Thus, the algorithm for computing the number $hcn(G)$ is also a recognition algorithm.

Let G be a non Hamiltonian QT-graph and let $T_c(G)$ be its cent-tree. From Theorem 3.1, we obtain that there are some nodes in the cent-tree $T_c(G)$ with negative H-labels and the h-tree of $T_c(G)$ does not exist; that is, Algorithm HT_CON fails to construct the h-tree $T_h(G)$. That means, we cannot achieve a one-to-one correspondence between the m-vertices and the dummy vertices of the available-tree of G; see Section 5.

In this case we are interested in computing the minimum number of edges which need to be added to $E(G)$ to make the graph G Hamiltonian. To this end, we construct a Hamiltonian QT-graph D from the graph G by adding a number of vertices in the set $V(G)$ and appropriate edges in the set $E(G)$; we call these vertices *d-vertices*. We shall define the graph D through its corresponding h-tree; we call dh-tree the h-tree of the graph D. The dh-tree is constructed from the cent-tree $T_c(G)$ of the graph G; the construction algorithm is as follows.

Algorithm DH-Tree-Construction (DHT_CON):

Step 1. **Modify** Step 6 of Algorithm HT_CON as follows:
"For $i = 1, 2, \ldots, \lceil \log n \rceil$ do, execute Steps 7 through 9";
Let T_a be the available-tree along with its dummy vertices, which is returned by the modified Algorithm HT_CON;

Step 2. **For** each m-vertex $u_i \in T_a$, do in parallel
2.1 Compute the number c_i of the dummy (white) children of vertex u_i;

Step 3. **For** each m-vertex $u_i \in T_a$ such that $c_i > 1$, do in parallel
3.1 Find the node V_i of the cent-tree $T_c(G)$ in which the m-vertex u_i belongs;
3.2 Add c_i d-vertices into node V_i;

end.

The time and processor complexity of the parallel algorithm for constructing the dh-tree of the graph D can be computed as follows: *Step 1*: The modified Algorithm HT_CON runs in $O(\log^2 n)$ time with

$O(n + m)$ processors on the CREW PRAM model. *Step 2*: It is easy to see that the available-tree contains $O(n)$ vertices, since it contains less vertices than the degree-tree $T_d(G)$, which contains n vertices, and at most n-1 dummy vertices. Thus, the number c_i of the dummy children of a vertex u_i is computed in $O(\log n_i)$ time with $O(n_i)$ processors on the EREW PRAM, where $1 \leq i < 2n$. Thus, this step is executed in $O(\log n)$ total time with $O(n)$ processors. *Step 3*: Both Substeps 3.1 and 3.2 can be executed in $O(1)$ with $O(n)$ processors on the EREW PRAM. Thus, we have the following result.

Lemma 7.1. Algorithm DHT_CON runs in $O(\log^2 n)$ time with $O(n + m)$ processors on the CREW PRAM model.

The *dh*-tree, constructed by Algorithm DHT_CON, and the cent-tree $T_c(G)$ have the same structures. Moreover, the *dh*-tree has the property that *H-label*$(V_i) \geq 0$ for each its node V_i, $1 \leq i \leq k$; note that the node V_i of the *dh*-tree contains the vertices of the node V_i of the cent-tree along with, probably, some other *d*-vertices. Thus, the graph D that corresponds to *dh*-tree is a Hamiltonian *QT*-graph.

More precisely, let $v_1, v_2, \ldots, v_n$ be the vertices of G and let $u'_1, u'_2, \ldots, u'_h$ be the *d*-vertices added to $T_c(G)$ by the *dh*-tree construction algorithm. Then, the graph D is the following:

(i) $V(D) = \{v_1, v_2, \ldots, v_n, u'_1, u'_2, \ldots, u'_h\}$, and

(ii) $E(D)$ contains all the edges of G and the edge $(u', v) \in E(D)$, if $u' \in \{u'_1, u'_2, \ldots, u'_h\}$ with $u' \in V_i$ and $v \in V_j$, and either $V_j = V_i$ or V_j is an ancestor or predecessor of V_i in the cent-tree $T_c(G)$

By construction, the Hamiltonian *QT*-graph D contains the *QT*-graph G as an induced subgraph. Moreover, it is easy to see that if we remove a vertex from D then it is no longer a Hamiltonian *QT*-graph. This proves the following result.

Lemma 7.2. The graph D is a minimum order Hamiltonian *QT*-graph that contains the *QT*-graph G as an induced subgraph.

Let G be a non Hamiltonian *QT*-graph and let $T_c(G)$ be its cent-tree. We consider the Hamiltonian *QT*-graph D along with its *dh*-tree. We have shown that, a Hamiltonian cycle in the graph D can be produced using the *h*-dfs tree of D; recall that the *h*-dfs tree is constructed by the *h*-dfs traversal strategy on the *h*-tree. We prove the following result.

Lemma 7.3. The Hamiltonian completion number $hcn(G)$ of a non Hamiltonian *QT*-graph G equals the number of *d*-vertices added to $V(G)$ by Algorithm DHT-CON; that is, $hcn(G) = |V(D)| - |V(G)|$.

Proof. Consider the *h*-sequence $(V_{f(1)}, V_{a(1)}, \ldots, V_1)$ of the *h*-tree and the *h*-dfs tree of the graph D. Let $v \in V_{f(1)}$ be the root of the *h*-dfs tree and let $HC = (v, \ldots, v_i, u', v_k, \ldots, v)$ be the Hamiltonian cycle which is produced by the *h*-dfs tree, where u' is a *d*-vertex. Note that, HC is a cycle on $n + h$ vertices, where n is the number of vertices in G and h is the number of *d*-vertices added by Algorithm DHT-CON in the nodes of the cent-tree $T_c(G)$. By construction, the cycle HC has the following properties:

(i) If u' is a *d*-vertex in HC, then its two adjacency vertices, say, v_i and v_k, are not *d*-vertices; that is, v_i, $v_k \in V(G)$, and

(ii) if $v_i \in V_i$ and $v_k \in V_k$, then both nodes V_i and V_k are leaves in $T_c(G)$ and $V_i \neq V_k$.

Thus, if we remove each d-vertex u' from HC and makes the vertices v_i and v_k to be adjacent, the resulting structure HC^* is a cycle on n vertices $v_1, v_2, \ldots, v_n$.

Since the vertices v_i and v_k belong to deferent leaves in $T_c(G)$, it follows that v_i and v_k are not adjacent in the graph G. Thus, if we add the edges (v_i, v_k) in $E(G)$, then the resulting graph G^* is Hamiltonian and the cycle HC^* is a Hamiltonian cycle of it.

We have proved that the number h of d-vertices that need to be added to the nodes of $T_c(G)$ to produce the dh-tree is minimum. The graph G^* is Hamiltonian, $V(G^*) = V(G)$, $E(G^*) \supset E(G)$ and $|E(G^*)| = |E(G)| + h$. Therefore, $h = hcn(G)$ and the lemma is proved. □

The preceding lemma provides a parallel algorithm for computing the Hamiltonian completion number of a QT-graph on n vertices. Moreover, it provides a parallel algorithm for computing the Hamiltonian completion edge set $CE(G)$; that is, the set of edge which need to be added to $E(G)$ to make G Hamiltonian. Note that, $|CE(G)| = hcn(G)$. Thus, we have the following theorem.

Theorem 7.1. The Hamiltonian completion number of a QT-graph G on n vertices and m edges can be computed in $O(\log^2 n)$ time with $O(n + m)$ processors on the CREW PRAM model. Moreover, the Hamiltonian completion edge set of G can be computed within the same time bound.

The algorithms we have previously described for computing the number $hcn(G)$ and constructing the edge set $CE(G)$ of a QT-graph are based on the computation of a Hamiltonian cycle of the graph D and construction of the d-tree of D.

It is possible, however, to obtain a much simpler parallel computation for the Hamiltonian completion number of a QT-graph G, if we make use of the structural properties of the cent-tree $T_c(G)$ and the H-labels of its nodes. This computation is described in the following algorithm; recall that this algorithm can also be served as a recognition algorithm for Hamiltonian QT-graphs.

Algorithm Hamiltonian-Completion-Number (HCN):

Step 1. **Compute** the cent-tree $T_c(G)$ of G using Algorithm CT_CON;
Let $V_1, V_2, \ldots, V_k$ be the nodes of the tree $T_c(G)$ and let $r_c = V_1$ be its root;

Step 2. **Make** the cent-tree $T_c(G)$ binary as follows:
Replace each node V_i having more than two children, say, $V_{i1}, V_{i2}, \ldots, V_{ip}$,
with a balanced binary tree whose leaves are the children of V_i;
the root of the balanced binary tree is the node V_i and its internal
nodes $V_{i(p+1)}, V_{i(p+2)}, \ldots, V_{ip'}$ are empty sets;
Let T_b be the resulting binary tree and let $V_1, V_2, \ldots, V_{k'}$ be its nodes, $k' \geq k$;

Step 3. **For** each node $V_i \in T_b$ $(1 \leq i \leq k')$, compute the label $H\text{-}label(V_i)$ as follows:
$H\text{-}label(V_i) \leftarrow |V_i| - 2$, if V_i is the root of the tree,
$H\text{-}label(V_i) \leftarrow |V_i| - 1$, if V_i is an internal node, and
$H\text{-}label(V_i) \leftarrow 0$, if V_i is a leaf;

Step 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 the H-label either of the node $p(p(V_i))$, that is, the parent of the node $p(V_i)$ or of the node $sib(V_i)$, as follows:

If $H\text{-}label(p(V_i)) < 0$ then

$H\text{-}label(p(p(V_i))) \leftarrow H\text{-}label(p(p(V_i))) + H\text{-}label(p(V_i))$;

else

if $sib(V_i)$ is an internal node then

$H\text{-}label(sib(V_i)) \leftarrow H\text{-}label(sib(V_i)) + H\text{-}label(p(V_i))$;

Let V_1 be the root of the resulting three-node tree and let V_{11}, V_{12} be the children of V_1;

Step 5. **Compute** the Hamiltonian completion number $hcn(G)$ of G as follows:
If $H\text{-}label(V_1) < 0$ then $hcn(G) \leftarrow | H\text{-}label(V_1) + H\text{-}label(V_{11}) + H\text{-}label(V_{12}) |$
else $hcn(G) \leftarrow 0$; that is, G is a Hamiltonian QT-graph;

end.

We next compute the time-processor complexity of the proposed parallel algorithm for recognizing Hamiltonian QT-graphs, using a step-by-step analysis.

Step 1: The cent-tree $T_c(G)$ of a quasi-threshold graph on n vertices and m edges is constructed in $O(\log n)$ time with $O(n + m)$ processors on the CREW PRAM model using Algorithm CT_CON.

Step 2: The cent-tree $T_c(G)$ can be made binary by replacing each node V_i having more than two children, with a balanced binary tree whose leaves are the children of V_i. It is easy to see that this computation can be done in $O(\log n)$ time with $O(n)$ processors on the EREW PRAM.

Step 3: The $H\text{-}label(V_i)$ of a node V_i is computed in $O(\log n_i)$ time with $O(n_i)$ processors on the EREW PRAM, where $1 \leq i \leq k'$ and $k' = O(n)$. Thus, this step is executed in $O(\log n)$ total time with $O(n)$ processors.

Step 4: It is well-known that we can shrink a binary tree into a three-node binary tree by successfully applying the rake operation which merging a leaf with its parent; the parallel tree-contraction algorithm can be implemented on the EREW PRAM model in $O(\log n)$ time using $O(n / \log n)$ processors [4, 20]. Since $k' = O(n)$, this step is executed in $O(\log n)$ time with $O(n / \log n)$ processors on the same computational model.

Step 5: Obviously, both steps are executed in $O(1)$ sequential time.

From the previous step-by-step analysis, we conclude that the recognition algorithm HCR 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 7.2. 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.

Corollary 7.1. 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.

8. 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 , ..., V_k be the nodes of the cent-tree T_c of G. We have shown in Section 2 that for every two nodes V_s and V_t such that $V_s \leq V_t$; that is, V_s is an ancestor of V_t in T_c, the graph $G[\{\cup V_i \mid V_s \leq V_i \leq V_t\}]$ is a complete graph. Moreover, for every maximal element V_t of $(\{V_i\}, \leq)$, the graph $G[\{\cup V_i \mid V_1 \leq V_i \leq V_t\}]$ is a maximal complete subgraph of G.

Based on these properties, it is easy to see that the height of the tree T_g of the graph G, which is constructed by the Algorithm QTR, equals the clique number $\omega(G)$ minus 1; recall that a leaf vertex of the tree T_g has height 0. Moreover, the vertex set which contains the vertices of the ith level of the tree T_g induces a stable graph. Since $\omega(G) = \chi(G)$, we can color the graph G by computing the level $l(v_i)$ of each vertex v_i of the tree T_g and setting $color(v_i) = l(v_i)$, $1 \leq i \leq n$; assuming that $l(r) = 1$, where r is the root of the tree T_g.

Let u be a leaf of the tree T_g such that $l(u) = \omega(G)$ and let MC be the set of vertices of the path from the root r of T_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_g.

Let $S = \{v_s, v_{s+1}, ..., v_t, ..., v_q\}$ be a stable set such that $v_t \in V_t$ and V_t is a maximal element of $(\{V_i\}, \leq)$ or, equivalently, V_t is a leaf node of T_c, $s \leq t \leq q$. 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 S is the set leaves vertices of the tree T_g. We have seen that we can find the leaves of the tree T_g using the Euler-tour technique on T_g.

Taking into consideration the above discussion, the complexity of the algorithms for constructing the trees T_c and T_g, and the complexity of some standard algorithmic techniques for computing the level function, the set of the leaves and certain paths on the tree T_g, 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.

9. 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, a tree structure, we called it *cent-tree*, which meets the structural properties of G.

By taking advantage of these properties and conditions, we constructed efficient parallel algorithms for finding a Hamiltonian cycle and computing the Hamiltonian completion number and the Hamiltonian completion edge set of a QT-graph; our algorithms run in $O(\log^2 n)$ 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. 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, which, in turn, implies an $O(\log n)$-time parallel algorithm for recognizing whether a QT-graph is Hamiltonian. 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.

Based on the structure of the cent-tree of a QT-graph, we also designed $O(\log n)$-time parallel algorithms for some other well-known optimization problems on QT-graphs. For example, we showed that the maximum clique problem, the maximum independent set problem, the clique cover problem and

the coloring problem can be solved in $O(\log n)$ time with 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 [34]). A second problem that is worth studying is the weighted version of the Hamiltonian completion edge set problem: we wish to minimize the total weight of the edges (with respect to the weights of its end-vertices) of the set $CH(G)$. 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 [7, 8, 29, 32]). 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.S. Adhar and S. Peng, Parallel algorithms for cographs and parity graphs with applications, *J. of Algorithms* 11 (1990) 252-284.

[2] A. Agnetis, P. Detti, C. Meloni and D. Pacciarelli, A linear algorithm for the Hamiltonian completion number of the line graph of a tree, *Inform. Process. Lett.* 79 (2001) 17-24.

[3] S. De Agostino and R. Petreschi, Parallel recognition algorithms for graphs with restricted neighbourhoods, *Inter. J. of Foundations of Computer Science* 1 (1990) 123-130.

[4] S. G. Akl, Parallel Computation: Models and Methods, Prentice-Hall, 1997.

[5] A.A. Bertossi, The edge Hamiltonian problem is NP-hard, *Inform. Process. Lett.* 13 (1981) 177-159.

[6] A. Brandstädt, B. Le and J.P. Spinrad, *Graph Classes: A Survey*, Siam Monographs on Discrete Math. and Appl., 1999.

[7] H.J. Broersma, Subgraphs conditions for dominating circuits in graphs and pancyclicity in line graphs, *Ars Combinatoria* 23 (1987) 5-12.

[8] R.A. Brualdi and R.F. Shong, Hamiltonian line graphs, *J. of Graph Theory* 5 (1981) 304-307.

[9] V. Chvatal and P.L. Hammer, Set-packing and threshold graphs, Res. Report CORR 73-21, University of Warerloo, 1973.

[10] D.G Corneil, H. Lerches and L. Burlingham, Complement reducible graphs, *Discrete Appl. Math.* 3 (1981) 163-174.

[11] D.G. Corneil, Y. Perl and L.K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput.* 14 (1985) 926-934.

[12] E. Dahlhaus, Efficient parallel recognition algorithms of cographs and distance hereditary graphs, *Discrete Appl. Math.* 57 (1995) 29-44.

[13] P. Detti and C. Meloni, A linear algorithm for the Hamiltonian completion number of the line graph of a cactus, *Proc. CTW'2001*, in: *Electronic Notes in Discrete Math.* (Elsevier), vol. 8, 2001.

[14] S. Goodman and S. Hedetniemi, On the Hamiltonian completion problems, in: A. Dold, B. Eckman (Eds), Graphs and Combinatorics, Lecture Notes in Math. (Springer, Berlin), vol. 406, pp. 262-272, 1974.

[15] M.C. Golumbic, Trivially perfect graphs, *Discrete Math.* 24 (1978) 105-107.

[16] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, Inc., New York, 1980.

[17] F. Harary, *Graph Theory*, Addison-Wesley, Reading, 1969.

[18] F. Harary and S.D. Nikolopoulos, On complete systems of invariants for small graphs, *Inter. J. of Comput. Math.* 64 (1997) 35-46.

[19] X. He, Parallel algorithms for cographs with applications, in: O. Nurmi and E. Ukkonen, eds., Algorithm Theory - SWAT '92, Lecture Notes in Comput. Sci. (Springer-Verlag), vol. 621, pp. 94-105, 1992.

[20] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, Inc., 1992.

[21] S. Kundu, A linear algorithm for the Hamiltonian completion number of a tree, *Inform. Process. Lett.* 5 (1976) 55-57.

[22] R. Lin and S. Olariu, An NC recognition algorithm for cographs, *J. of Parallel and Distrib. Comput.* 13 (1991) 76-90.

[23] S. Ma, W.D. Wallis and J. Wu, Optimization problems on quasi-threshold graphs, *J. Comb. Inform. & Syst. Sciences*. 14 (1989) 105-110.

[24] T.A. McKee and F. R. McMorris, *Topics in Intersection Graph Theory*, Siam Monographs on Discrete Math. and Appl., 1999.

[25] S.D. Nikolopoulos, Constant-time parallel recognition of split graphs, *Inform. Process. Lett.* 54 (1995) 1-8.

[26] S.D. Nikolopoulos, Recognizing cographs and threshold graphs through a classification of their edges, *Inform. Process. Lett.* 75 (2000) 129-139.

[27] A. Raychaudhuri, The total interval number of a tree and the Hamiltonian completion number of its line graph, *Inform. Process. Lett.* 56 (1995) 299-306.

[28] J. Reif (editor), *Synthesis of Parallel Algorithms*, Morgan Kaufmann, Inc., 1993.

[29] H.J. Veldman, A result on Hamiltonian line graphs involving restrictions on induced subgraphs, *J. of Graph Theory* 12 (1988) 413-420.

[30] E.S. Wolk, The comparability graph of a tree, *Proc. Amer. Math, Soc.* 3 (1962) 789-795.

[31] E.S. Wolk, A note of the comparability graph of a tree, *Proc. Amer. Math, Soc.* 16 (1965) 17-20.

[32] L. Xiong, H.J. Broersma, C. Hoede and X. Li, Degree sum and subpancyclicity in line graphs, *Discrete Math.*, to appear.

[33] J-H. Yan, J-J. Chen and G.J. Chang, Quasi-threshold graphs, *Discrete Appl. Math.* 69 (1996) 147-255.

[34] S. Zhang, X. Li and H.J. Broersma, Heavy paths and cycles in weighted graphs, *Discrete Math.* 223 (2000) 327-336.