Parallel Algorithms for Recognizing P_5-free and $\overline{P}_5$-free Weakly Chordal Graphs

Stavros D. Nikolopoulos and Leonidas Palios

Department of Computer Science, University of Ioannina
P.O.Box 1186, GR-45110 Ioannina, Greece
e-mail: {stavros, palios}@cs.uoi.gr

Abstract: We prove algorithmic characterizations of weakly chordal graphs, which lead to efficient parallel algorithms for recognizing P_5-free and $\overline{P}_5$-free weakly chordal graphs. For an input graph on n vertices and m edges, our algorithms run in $O(\log^2 n)$ time and require $O(m^2/\log n)$ processors on the EREW PRAM model of computation. The proposed recognition algorithms efficiently detect P_5s and $\overline{P}_5$s in weakly chordal graphs in $O(\log n)$ time with $O(m^2/\log n)$ processors on the EREW PRAM. Additionally, we show how the algorithms can be augmented to provide a certificate for the existence of a P_5 (or a $\overline{P}_5$) in case the input graph is not P_5-free (respectively, $\overline{P}_5$-free) weakly chordal.

Keywords: Parallel algorithm, recognition, weakly chordal graph, chordless path.

1 Introduction

Let G be an undirected graph with no loops or multiple edges, and let C (resp. P) be a simple cycle (resp. path) in G. A *chord* of the cycle C (path P) is an edge of G incident on two non-consecutive vertices of C (P). A *chordless cycle* (*chordless path*) is a simple cycle (path) containing no chords. A chordless cycle on k vertices (or equivalently, a chordless cycle of length k) is denoted C_k; a chordless path on k vertices is denoted P_k.

An undirected graph G is called *chordal* (or *triangulated*) if it has no chordless cycle of length greater than or equal to 4 (see Golumbic [10]); it is called *weakly chordal* if both G and its complement $\overline{G}$ have no chordless cycle of length greater than or equal to 5 (see Hayward [11]). A weakly chordal graph is called P_5*-free* ($\overline{P}_5$*-free*) *weakly chordal* if it has no induced subgraph isomorphic to P_5 ($\overline{P}_5$ respectively).

Weakly chordal graphs were introduced by Hayward [11] as a natural extension of the well-known class of chordal graphs. Chordal graphs arise in the study of Gaussian elimination on sparse symmetric matrices [22], in the study of acyclic relational schemes [3], and are related to and useful for many location problems (see Golumbic [10]). Due to a result of Chvátal [7], which states that a linear order "$<$" is *perfect* iff there is no chordless path $abcd$ with $a < b$ and $d < c$, it can be shown that the chordal graphs are *perfectly orderable*, that is, they admit a perfect order. Hayward [11] proved that weakly chordal graphs are perfect, but not all weakly chordal graphs are perfectly orderable; in fact, Hoáng has shown that determining whether a graph is perfectly orderable remains NP-complete for the class of weakly chordal graphs [17]. In 1990, Chvátal conjectured that every P_5-free weakly chordal graph is perfectly orderable [8]. Recently, Hayward [13] proved the conjecture by presenting a polynomial time algorithm to find a perfect order of any P_5-free weakly chordal graph. On the other hand, it is known that the $\overline{P}_5$-free weakly chordal graphs are not necessarily perfectly orderable [13].

Authors	Time	Processors	Model
Hayward [12]	$O(n^5)$	-	-
Spinrad [23]	$O(n^{4.376})$	-	-
Spinrad and Sritharan [24]	$O(n^2m)$	-	-
Hayward *et al.* [15]	$O(m^2)$	-	-
Berry *et al.* [4]	$O(m^2)$	-	-
Nikolopoulos and Palios [21]	$O(m^2)$	-	-
Chandrasekharan *et al.* [5]	$O(\log n)$	$O(n^5)$	CRCW
Chong, Nikolopoulos, and Palios [6]	$O(\log^2 n)$	$O(m^2/\log n)$	EREW

Table 1: Sequential and parallel algorithms for recognizing weakly chordal graphs.

The problem of recognizing weakly chordal graphs has been extensively studied, mainly in the context of finding chordless cycles of length greater than or equal to 5. Hayward [12] proposed an $O(n^k)$-time sequential algorithm for detecting chordless cycles of length greater than or equal to k. This algorithm can be used to recognize whether a graph G is weakly chordal by checking for the presence of chordless cycles of length greater than or equal to 5 in G and then in its complement $\overline{G}$; this in turn leads to an $O(n^5)$-time recognition algorithm for weakly chordal graphs. Hayward's result was improved to $O(n^{4.376})$ by Spinrad's hole-finding procedure [23] implying an algorithm for recognizing weakly chordal graphs of the same time complexity. Arikati and Rangan [2] gave an efficient algorithm for finding 2-pairs on a graph (a pair of non-adjacent vertices, say, x and y, such that every chordless path from x to y has exactly two edges [14]), and Spinrad and Sritharan [24] used this to provide a sequential algorithm for recognizing weakly chordal graphs in $O(n^2m)$ time. Hayward [13] introduced the notion of the so-called *handle* (a separable set of edges), which was used recently by Hayward, Spinrad, and Sritharan [15] to give an $O(m^2)$-time recognition algorithm; their algorithm finds a set of co-pairs by computing a handle of a handle recursively, and by repeatedly removing a co-pair, until no co-pair is left in the graph. More recently, Berry *et al.* [4] proposed an algorithm for recognizing weakly chordal graphs that is not based on the notion of a 2-pair or a co-pair; their algorithm matches the $O(m^2)$-time complexity of the algorithm of [15] but it requires more space, $O(m^2)$, compared to $O(m)$ of [15]. Finally, the hole and antihole detection algorithms of Nikolopoulos and Palios [21] can be used to yield an $O(m^2)$-time weakly chordal graph recognition algorithm.

Although chordal graphs have been the focus of much research in both sequential and parallel process environment, weakly chordal graphs have not received as much attention in the parallel environment, despite the fact that the definitions of the chordal and the weakly chordal graphs rely on the same principle.

The results of Hayward [12] imply a parallel recognition algorithm for weakly chordal graphs running in $O(\log n)$ time with $O(n^5)$ processors on a CRCW PRAM. On the other hand, the weakly chordal graph recognition algorithm proposed by Spinrad and Sritharan [24] does not seem to be amenable to parallelization. Chandrasekharan *et al.* [5] presented a parallel algorithm for obtaining a chordless cycle of length greater than or equal to $k \geq 4$ in a graph in $O(n^{k-4}m^2)$ time sequentially and in $O(\log n)$ time using $O(n^{k-4}m^2)$ processors in parallel on the CRCW PRAM, whenever such a cycle exists. By setting $k = 4$, we see that a chordless cycle of length greater than or equal to 4 can be found in $O(\log n)$ time using $O(m^2)$ processors, while by setting $k = 5$, a chordless cycle of length greater than or equal to 5 can be found in $O(\log n)$ time using $O(nm^2)$ processors. These results lead to parallel algorithms for recognizing chordal and weakly chordal graphs running in $O(\log n)$ time on a CRCW PRAM using $O(n^4)$ and $O(n^5)$ processors, respectively. Recently, Chong, Nikolopoulos, and Palios [6] described a parallel version of the weakly chordal graph recognition algorithm of Berry *et al.* which runs in $O(\log^2 n)$ time using $O(m^2/\log n)$ processors on the EREW PRAM. Table 1 summarizes the results on sequential and parallel algorithms for the recognition of weakly chordal graphs.

In this paper, we present efficient parallel algorithms for the recognition of P_5-free and $\overline{P}_5$-free

weakly chordal graphs. We prove characterizations of weakly chordal graphs, which enable us to efficiently detect P_5s and $\overline{P}_5$s in them. In particular, we can detect P_5s and $\overline{P}_5$s in weakly chordal graphs in $O(\log n)$ time with $O(m^2/\log n)$ processors on the EREW PRAM. The combination of these algorithms with the efficient parallel algorithm of [6] for recognizing weakly chordal graphs results in algorithms to recognize P_5-free and $\overline{P}_5$-free weakly chordal graphs, which run in $O(\log^2 n)$ time using $O(m^2/\log n)$ processors on the EREW PRAM computational model. Additionally, we show how the algorithms can be augmented to provide a certificate for the existence of a P_5 (or a $\overline{P}_5$) in case the input graph is not P_5-free (respectively, $\overline{P}_5$-free) weakly chordal.

The paper is organized as follows. In Section 2 we present the notation and related terminology and we prove conditions for a weakly chordal graph to contain a P_5 or a $\overline{P}_5$ as an induced subgraph. In Section 3 we present the parallel recognition algorithms and analyze their time and processor complexities. Finally, in Section 4 we conclude the paper and discuss possible future extensions.

2 Definitions and Characterizations

We consider finite undirected graphs with no loops or multiple edges. Let G be such a graph; we denote the vertex set of G by $V(G)$ and its edge set by $E(G)$. 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 $G - S := G[V(G) - S]$.

A *chord* is an edge between two non-consecutive vertices of a cycle or a path. A *hole* is an induced chordless cycle on five or more vertices, and an *antihole* is the complement of a hole. In light of the definition of a hole and an antihole, the weakly chordal graphs are defined as the graphs that contain no hole or antihole.

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 $N[x] := \{x\} \cup N(x)$. The neighborhood of a subset A of vertices is defined as $N(A) := \left(\bigcup_{x \in A} N(x)\right) - A$ and its closed neighborhood as $N[A] := A \cup N(A)$. If $e = xy$ is an edge of G, then we use $N(e)$ (resp. $N[e]$) to denote the vertex set $N(\{x, y\})$ (resp. $N[\{x, y\}]$); we call the vertex sets $N(e)$ and $N[e]$ neighborhood and closed neighborhood of the edge e, respectively. For an edge $e = xy$, we consider the following three sets:

$$A(x; e) = N(x) - N[y],$$
$$A(y; e) = N(y) - N[x],$$
$$A(e) = N(x) \cap N(y).$$

Clearly, these sets partition the neighborhood $N(e)$ of the edge e.

We next provide characterizations of P_5-free and $\overline{P}_5$-free weakly chordal graphs. In order to simplify the notation, we use the following definitions:

Definition 1. Let $e = xy$ be an edge of a weakly chordal graph G. Then, the edge e of G is said to be a *P_5-witness* if there exist vertices $a \in A(x; e)$, $b \in A(y; e)$, and $u \in V(G) - N[e]$ such that $ab \notin E(G)$ and $ua \in E(G)$. (See Figure 1.)

Definition 2. Let e be an edge of a weakly chordal graph G. Then, the edge e is said to be a *$\overline{P}_5$-witness* if there exists a vertex $u \in V(G) - N[e]$ such that $N(u)$ contains vertices from both $A(x; e)$ and $A(y; e)$. (See Figure 2.)

We prove the following results.

Lemma 2.1. *Let G be a weakly chordal graph. Then, G is P_5-free weakly chordal if and only if none of its edges is a P_5-witness.*

Proof: ($\Longrightarrow$) Let G be a P_5-free weakly chordal graph, and suppose, for contradiction, that there exists an edge $e = xy$ of G which is a P_5-witness; this implies that there exist vertices $a \in A(x; e)$,

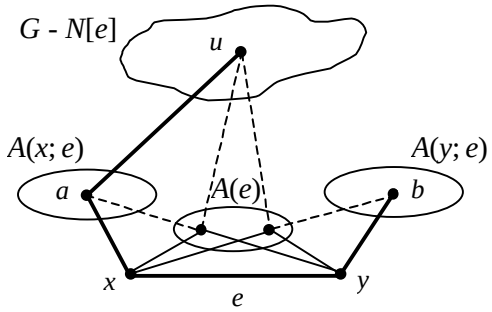

Figure 1

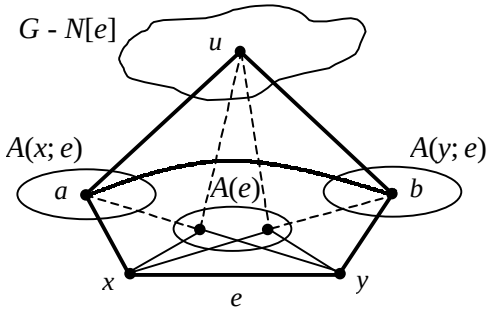

Figure 2

$b \in A(y;e)$, and $u \in V(G) - N[e]$ such that $ab \notin E(G)$ and $ua \in E(G)$. Because G is weakly chordal, $ub \notin E(G)$, for otherwise G would contain a C_5. Then, the path $uaxyb$ is a P_5 in G, a contradiction to the fact that G is a P_5-free weakly chordal graph. Therefore, no edge of G is a P_5-witness.

($\Longleftarrow$) Suppose that none of the edges of G is a P_5-witness. Then, G is a P_5-free weakly chordal graph. If not, G would contain a P_5 as an induced subgraph; let this P_5 be $pqrst$. But then, the edge $e = rs$ would be a P_5-witness; note that the vertices p, q, t meet the conditions of Definition 1 for the vertices u, a, b, respectively. This is a contradiction; hence, G is a P_5-free weakly chordal graph. ∎

Lemma 2.2. *Let G be a weakly chordal graph. Then, G is $\overline{P}_5$-free weakly chordal if and only if none of its edges is a $\overline{P}_5$-witness.*

Proof: ($\Longrightarrow$) Let G be a $\overline{P}_5$-free weakly chordal graph, and suppose, for contradiction, that there exists an edge $e = xy$ of G which is a $\overline{P}_5$-witness. This implies that there exists a vertex u such that $u \in V(G) - N[e]$ and $N(u)$ contains vertices from both $A(x;e)$ and $A(y;e)$; let $a, b \in N(u)$ and $a \in A(x;e)$ and $b \in A(y;e)$ (see Figure 2). The vertices a and b are adjacent in G; otherwise, the vertices u, a, x, y, b induce a C_5 in G, in contradiction to the fact that G is a weakly chordal graph. Then, the subgraph of G induced by x, y, a, b, u is a $\overline{P}_5$ in G, a contradiction to the fact that G is $\overline{P}_5$-free. Therefore, none of G's edges is a $\overline{P}_5$-witness.

($\Longleftarrow$) Suppose that none of the edges of G is a $\overline{P}_5$-witness. If G is not $\overline{P}_5$-free, G contains a $\overline{P}_5$ as an induced subgraph; let this subgraph be the complement of the P_5 $pqrst$. Let us consider the edge $e = qs$ of G. Clearly, $r \in V(G) - N[e]$ and it is adjacent to both p and t, where $p \in A(q;e)$ and $t \in A(s;e)$. Hence, the edge e is a $\overline{P}_5$-witness; a contradiction. Therefore, G is a $\overline{P}_5$-free weakly chordal graph. ∎

3 Recognizing P_5-free and $\overline{P}_5$-free Weakly Chordal Graphs

In this section we present parallel algorithms for the recognition of P_5-free and $\overline{P}_5$-free weakly chordal graphs. The algorithms are based on the notions of a P_5-witness and a $\overline{P}_5$-witness: they process each edge of the input graph to verify whether it is a P_5-witness (or a $\overline{P}_5$-witness); if any of them is so, then the graph is not a P_5-free ($\overline{P}_5$-free respectively) weakly chordal graph, otherwise it is. This is a direct application of Lemmas 2.1 and 2.2. However, for Lemmas 2.1 and 2.2 to be applicable, the graph in question must be a weakly chordal graph. To ensure that, our algorithms employ at a preprocessing step the parallel algorithm described in [6] for recognizing weakly chordal graphs; the algorithm is a parallel version of the weakly chordal graph recognition algorithm of Berry *et al.* [4] and relies on an optimal parallel co-connectivity algorithm. Its complexity is summarized in the following theorem.

Theorem 3.1 ([6]). *Weakly chordal graphs can be recognized in $O(\log^2 n)$ time using $O(m^2/\log n)$ processors on the EREW PRAM model, where n and m are the numbers of vertices and edges of the input graph respectively.*

3.1 Recognizing P_5-free Weakly Chordal Graphs

We next describe a parallel algorithm which takes as input a weakly chordal graph G and determines whether it is P_5-free. The graph G is assumed to be given in its adjacency list representation. The algorithm checks if there exists an edge in G which is a P_5-witness; if no such edge exists, then the graph is P_5-free, otherwise it is not (Lemma 2.1). The algorithm operates as follows:

Algorithm P_5-FREE-WC_REC:

1. construct an array $E[\,]$ of size $2m$, which, for each edge uw of the input graph G, contains both the *twin* pairs (u,w) and (w,u): the contents of the array are so that all the pairs with the same first element occupy consecutive positions in the array, and additionally each pair (u,w) is linked to its twin (w,u);
   make m copies of the array $E[\,]$, and associate with each edge e of G one such copy, say, $E_e[\,]$;

2. For each edge $e = xy$ of the graph G do in parallel

   2.1 compute the vertex sets $A(x;e)$, $A(y;e)$, and $V(G) - N[e]$, and the cardinalities $|A(x;e)|$ and $|A(y;e)|$;

   2.2 for each vertex a in $A(x;e)$ do in parallel
   count the neighbors of a which belong to $A(y;e)$;
   if their number is less than $|A(y;e)|$
   then mark the vertex a;

   2.3 for each vertex b in $A(y;e)$ do in parallel
   count the neighbors of b which belong to $A(x;e)$;
   if their number is less than $|A(x;e)|$
   then mark the vertex b;

   2.4 $M[e] \leftarrow 0$; $\{M[\,]$ *is an auxiliary array of size* $m\}$
   for each vertex $u \in V(G) - N[e]$ do in parallel
   if any of the neighbors of u is marked
   then $M[e] \leftarrow 1$;

3. If there exists an edge e such that $M[e] = 1$, then the edge e is a P_5-witness, and, thus, G is not a P_5-free weakly chordal graph; otherwise, G is a P_5-free weakly chordal graph;

Correctness. The correctness of the algorithm follows from Lemma 2.1, taking into account that if for a vertex $a \in A(x;e)$ it holds that $|N(a) \cap A(y;e)| < |A(y;e)|$, then there exists a vertex in $A(y;e)$ which is not adjacent to a in G.

Time and Processor Complexity. Let us now analyze the time and processor complexity of the algorithm. We assume that the input graph is connected (otherwise, we work on its connected components); thus, $\log m = \Theta(\log n)$. We compute the complexity of each step separately. For details on the PRAM techniques mentioned below, see [1] or [19].

Step 1. The array $E[\,]$ is built as follows: We compute the ranks of the elements in each of the adjacency lists. The largest rank in each adjacency list is its size; we collect these sizes in an auxiliary array of size n and we compute parallel prefix sums on it. Then, for each vertex v_i $(i = 1, \ldots, n)$ of the input graph G, we store the pairs (v_i, v_j), for all neighbors v_j of v_i, in the entries $E[ps[i-1]+1], \ldots, E[ps[i]]$, where $ps[k]$ denotes the k-th prefix sum and $ps[0] = 0$. To do that, we copy the neighbors of each vertex v_i to an auxiliary array, and we pass the values of v_i and $ps[i-1]$ to them by applying interval broadcasting on the array; then, if the j-th record of the adjacency list of u_i corresponds to the vertex v_k, the $(ps[i-1]+j)$-th entry of the array $E[\,]$ is set equal to (v_i, v_k). The above operations can be executed in $O(\log m)$ total time using $O(m/\log m)$ processors on the EREW PRAM model. The linking of the pairs (v_i, v_j) and (v_j, v_i), for each edge v_iv_j of G, is achieved in two phases by means of

a $\Theta(n^2)$-size array $A[]$: in the first phase, for each entry $E[k] = (v_i, v_j)$ the entry $A[i, j]$ is set equal to k; in the second phase, for each entry $E[k] = (v_i, v_j)$ the content of $A[j, i]$ is read; this is equal to the position of the pair (v_j, v_i) in the array $E[]$. The linking takes $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model. Finally, making m copies of the array $E[]$ takes $O(\log m)$ time using $O(m^2/\log m)$ processors on the EREW PRAM model.

Step 2. This step is executed for each one of the m edges of the graph G.

Substep 2.1: The vertex set $A(x; e)$ is stored in an array $A_{e,x}[]$ of size n. The entries of the array are initialized to 0; next, for each entry (x, v) of the array $E_e[]$, the entry $A_{e,x}[v]$ is set to 1; finally, for each entry (y, v) of the array $E_e[]$, the entry $A_{e,x}[v]$ is set to 0, and so is $A_{e,x}[y]$. The updating of the array $A_{e,x}[]$ takes $O(1)$ time using $O(m)$ processors, or $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model. An array $A_{e,y}[]$ storing the set $A(y; e)$ is computed similarly. An array $R_e[]$ storing the set $V(G) - N[e]$ is also computed in a similar fashion: its entries are initialized to 1; next, for each entry (x, v) of the array $E_e[]$, the entry $R_e[v]$ is set to 0; finally, for each entry (y, v) of the array $E_e[]$, the entry $R_e[v]$ is set to 0. The cardinalities of $A(x; e)$ and $A(y; e)$ can be computed by summing the entries of the arrays $A_{e,x}[]$ and $A_{e,y}[]$; this takes $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

Substep 2.2: Each vertex p uses interval broadcasting to pass to all the pairs (p, q) of the array $E_e[]$ a value which is equal to 1 if $p \in A(y; e)$, and 0 otherwise. Next, these values are passed to and stored at the twin pairs so that each pair (u, v) stores 1 if $v \in A(y; e)$, and 0 otherwise. Then, the number of neighbors of vertex u which belong to $A(y; e)$ is computed by summing the values of 0 or 1 stored at all the pairs (u, v). The above computations for all vertices of G take $O(\log m)$ time using $O(m/\log m)$ processors on the EREW PRAM. Checking whether the resulting numbers are equal to $|A(y; e)|$ takes $O(1)$ time and $O(n)$ processors, or $O(\log n)$ time and $O(n/\log n)$ processors, on the EREW PRAM assuming that each vertex has its own copy of $|A(y; e)|$, something which can be easily achieved in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.

Substep 2.3: Similarly to Substep 2.2, this substep can be executed in $O(\log m)$ time using $O(m/\log m)$ processors on the EREW PRAM model.

Substep 2.4: This substep too is similar to Substep 2.2. Each vertex p uses interval broadcasting to pass to all the pairs (p, q) of the array $E_e[]$ a value which is equal to 1 if p is marked, and 0 otherwise. Next, these values are passed to and stored at the twin pairs, and finally, for each vertex u, the values stored at all the pairs (u, v) are summed. These computations take $O(\log m)$ time using $O(m/\log m)$ processors on the EREW PRAM model. If the sum is not 0 for any vertex in $V(G) - N[e]$, then $M[e]$ needs to be set to 1. Directly setting the entry $M[e]$ results in concurrent writing; instead, we mark each such vertex u and we set $M[e] \leftarrow 1$ if and only if there exists a vertex in $V(G) - N[e]$ which has been marked (this is determined by computing the disjunction of the values of the predicate "belongs to $V(G) - N[e]$ and has been marked" for the vertices of G, and takes $O(\log n)$ time and $O(n/\log n)$ processors on the EREW PRAM).

Thus, in total, Step 2 is executed in $O(\log n)$ time with $O(m^2/\log n)$ processors on the EREW PRAM model.

Step 3. This step can be executed in $O(\log m)$ time and $O(m/\log m)$ processors on the EREW PRAM model by computing the maximum among the m entries of the array $M[]$ and testing whether it is equal to 1.

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

Theorem 3.2. *Given a weakly chordal graph on n vertices and m edges, it can be determined whether it is a P_5-free weakly chordal graph in $O(\log n)$ time using $O(m^2/\log n)$ processors on the EREW PRAM model.*

The algorithm P_5-FREE-WC_REC assumes that the input graph is weakly chordal; so, if we combine the parallel weakly chordal graph recognition algorithm of [6] with Algorithm P_5-FREE-WC_REC, we obtain an algorithm to determine whether a given graph is P_5-free weakly chordal. This is summarized in the following corollary:

Corollary 3.1. *P_5-free weakly chordal graphs can be recognized in $O(\log^2 n)$ time using $O(m^2/\log n)$ processors on the EREW PRAM model, where n and m are the numbers of vertices and edges of the input graph respectively.*

Certification of the Existence of a P_5. The algorithm can be easily augmented to output a certificate whenever it decides that the input weakly chordal graph is not P_5-free. Indeed, in Substep 2.4, if, for an edge e and a vertex u in $V(G) - N[e]$, there exists a neighbor of u which is marked, then we assign to $M[e]$ a positive integer uniquely coding the index numbers of u and its marked neighbor. Subsequently, in Step 3, if there exists an edge e' such that $M[e'] \neq 0$, then the algorithm reports the edge e' and the two vertices encoded in $M[e']$, from which the fifth vertex of the P_5 can be easily extracted.

3.2 Recognizing $\overline{P}_5$-free Weakly Chordal Graphs

We next describe a parallel algorithm which takes as input a weakly chordal graph G and determines whether it is $\overline{P}_5$-free. Again, the graph G is given in its adjacency list representation. The algorithm checks if there exists an edge in G which is a $\overline{P}_5$-witness; if there exists, then the graph is not $\overline{P}_5$-free, otherwise it is (Lemma 2.2). The algorithm operates as follows:

Algorithm $\overline{P}_5$-FREE-WC_REC:

1. construct an array $E[\,]$ of size $2m$, which, for each edge uw of the input graph G, contains both pairs (u, w) and (w, u): the contents of the array are so that all the pairs with the same first element occupy consecutive positions in the array, and additionally, for each edge uw, the pairs (u, w) and (w, u) are linked to each other;
   make m copies of the array $E[\,]$, and associate with each edge e of G one such copy, say, $E_e[\,]$;

2. For each edge $e = xy$ of the graph G, do in parallel

   2.1 for each entry $E_e[i] = (u_i, w_i)$, $1 \leq i \leq 2m$, do in parallel
   - (i) if $u_i \in V(G) - N[e]$ and $w_i \in A(x; e)$ then $E_e[i] \leftarrow (u_i, 1)$;
   - (ii) if $u_i \in V(G) - N[e]$ and $w_i \in A(y; e)$ then $E_e[i] \leftarrow (u_i, 2)$;
   - (iii) in all other cases, $E_e[i] \leftarrow (u_i, 0)$;

   2.2 remove from the array $E_e[\,]$ the entries $(*, 0)$ and then pack the remaining entries into consecutive locations;

   2.3 for each entry $E_e[i] = (u_i, \ell_i)$ do in parallel
   if $\ell_i = 1$ and either $E_e[i-1]$ or $E_e[i+1]$ is equal to $(u_i, 2)$ then set $E_e[i] \leftarrow (u_i, 3)$;

   2.4 compute the maximum over the second field of the entries of the array $E_e[\,]$; if it is equal to 3, then set $M[e] \leftarrow 1$, else set $M[e] \leftarrow 0$ ($M[\,]$ is an auxiliary array of size m);

3. If there exists an edge e of G such that $M[e] = 1$, then the edge e is a $\overline{P}_5$-witness, and, thus, G is not a $\overline{P}_5$-free weakly chordal graph; otherwise, G is a $\overline{P}_5$-free weakly chordal graph;

Correctness. Note that after Step 2.1, an entry $(u,1)$ (or $(u,2)$) in $E_e[\,]$ implies that u belongs to $V(G)-N[e]$ and is adjacent to a vertex in $A(x;e)$ ($A(y;e)$ respectively). The array $E_e[\,]$ is built so that pairs with the same first element are stored in consecutive positions; this property continues to hold after the completion of Step 2.2, as the array packing maintains the relative positions of the remaining entries. This implies that if a vertex $u \in V(G) - N[e]$ is adjacent to both a vertex in $A(x;e)$ and a vertex in $A(y;e)$, then after Step 2.2, the array $E_e[\,]$ contains both an entry $(u,1)$ and an entry $(u,2)$ among entries with first element u and second element 1 or 2. But then, at least a pair of entries $(u,1)$ and $(u,2)$ are located in consecutive positions, which in Step 2.3 will leave an entry $(u,3)$ in the array. The correctness follows from Lemma 2.2.

Time and Processor Complexity. Next we present the complexity analysis by discussing each step separately. Again, we assume that the input graph is connected, which implies that $\log m = \Theta(\log n)$.

Step 1. This step is identical to Step 1 of Algorithm P_5-FREE-WC_REC; it can be executed in $O(\log m)$ time and $O(m^2/\log m)$ processors on the EREW PRAM model.

Step 2. This step is executed for each one of the m edges of the graph G.

Substep 2.1: We use an auxiliary array $V_e[\,]$ of size n, which we update so that $V_e[v] = 1$ if $v \in A(x;e)$, $V_e[v] = 2$ if $v \in A(y;e)$, $V_e[v] = 3$ if $v \in V(G) - N[e]$, and $V_e[v] = 0$ in all other cases. In order to assign the correct values in $V_e[\,]$, we do the following for each vertex v of G: We use $O(\text{degree}(v))$ processors, one at each of the pairs (v,v'), where v' is a neighbor of v. Each such processor checks if the corresponding vertex v' is x, y, or none of them; in the former case, it produces a 1, in the second a 2, or otherwise a 0. Next, by using a tree-shape structure of processors, we compute the sum of the numbers produced by the processors at all these pairs (v,v'): if it is 1 and $v \neq y$, then we set $V_e[v]$ to 1; if it is 2 and $v \neq x$, then we set $V_e[v]$ to 2; if it is 0, then we set $V_e[v]$ to 3; in all other cases, we set $V_e[v]$ to 0. The updating of the array $V_e[\,]$ can be completed in $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model. Then, Substep 2.1 can be executed in $O(1)$ time on the EREW PRAM as soon as each entry $E_e[i] = (u_i, w_i)$ obtains its own copy of $V_e[u_i]$ and $V_e[w_i]$. The latter is achieved as follows: for each vertex v of G, the value of $V_e[v]$ is passed to and stored at all the pairs (v,v') in $E_e[\,]$ by means of interval broadcasting (recall that all such pairs occupy consecutive entries in $E_e[\,]$); next, each pair (u,w) in $E_e[\,]$ obtains $V_e[w]$ from the entry storing (w,u) to which it has a link. The above implementation can be executed in $O(\log m)$ time with $O(m/\log m)$ processors on the EREW PRAM model.

Substep 2.2: The removal of the $(*,0)$ entries from the array $E_e[\,]$ can be done by marking these entries with 0 and the remaining entries with 1 and then by using array packing on the array. This can be done in $O(\log m)$ time using $O(m/\log m)$ processors on the EREW PRAM model.

Substep 2.3: This step takes $O(1)$ time using $O(m)$ processors, or $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model; obtaining the values of $E_e[i]$, $E_e[i-1]$, and $E_e[i+1]$, for each i, needs to be done in three separate phases in order to be implemented using exclusive read operations.

Substep 2.4: Computing the maximum on an array of size m can be done in $O(\log m)$ time using $O(m/\log m)$ processors on the EREW PRAM; checking whether it is equal to 3 and assigning the correct value to $M[e]$ takes one processor and constant time.

Step 3. This step is identical to Step 3 of Algorithm P_5-FREE-WC_REC; it can be executed in $O(\log m)$ time and $O(m/\log m)$ processors on the EREW PRAM model.

Therefore, we have:

Theorem 3.3. *Given a weakly chordal graph on n vertices and m edges, it can be determined whether it is a $\overline{P}_5$-free weakly chordal graph in $O(\log n)$ time using $O(m^2/\log n)$ processors on the EREW PRAM model.*

Finally, combining Algorithm $\overline{P}_5$-FREE-WC_REC with the parallel weakly chordal graph recognition algorithm of [6], we have:

Corollary 3.2. *$\overline{P}_5$-free weakly chordal graphs can be recognized in $O(\log^2 n)$ time using $O(m^2/\log n)$ processors on the EREW PRAM model, where n and m are the numbers of vertices and edges of the input graph respectively.*

Corollaries 3.1 and 3.2 lead to the following result.

Corollary 3.3. *(P_5, $\overline{P}_5$)-free weakly chordal graphs can be recognized in $O(\log^2 n)$ time using a total of $O(m^2/\log n)$ processors on the EREW PRAM model, where n and m are the numbers of vertices and edges of the input graph respectively.*

Certification of the Existence of a $\overline{P}_5$. As with the previous algorithm, this one can be augmented so that, if the input graph is not P_5-free, it outputs a certificate (e, u), where $e = xy$ is an edge of the input graph and u is a vertex adjacent to vertices from both $A(x; e)$ and $A(y; e)$. What we need to do is to locate an entry $(u_i, 3)$ of the array $E_e[\,]$, whenever the maximum in Step 2.4 is found equal to 3; then, u is precisely u_i. In particular, Step 2.4 is augmented as follows: while computing the maximum, we also maintain a pair (u_i, ℓ_i) with a maximum value of ℓ_i. In this way, if the resulting pair is (u_j, ℓ_j) and ℓ_j is equal to 1, then we encode u_j in $M[e]$; u_j is precisely the sought u for the edge e. Subsequently, in Step 3, if there exists an edge e' such that $M[e'] \neq 0$, then the algorithm reports the edge e' and the vertex encoded in $M[e']$, from which the remaining two vertices of the $\overline{P}_5$ can be easily extracted.

4 Concluding Remarks

In this paper we present efficient parallel algorithms for recognizing P_5-free weakly chordal graphs, a subclass of perfectly orderable graphs, and $\overline{P}_5$-free weakly chordal graphs. Our algorithms run in $O(\log^2 n)$ time using $O(m^2/\log n)$ processors on the EREW PRAM model of computation, where n is the number of vertices and m is the number of edges of the input graph. It is worth noting that our algorithms can be easily augmented to provide a certificate for the existence of a P_5 (or a $\overline{P}_5$) in the input graph.

As mentioned in the introduction, Hayward [13] proved Chvatal's conjecture that every P_5-free weakly chordal graph is perfectly orderable [8]. This result implies that various subclasses of P_5-free weakly chordal graphs are now known to be perfectly orderable, namely,

- P_4-free graphs (they are also known as cographs) [20],
- (bull, P_5)-free weakly chordal graphs [8], where the *bull* is a graph with vertex set $\{a, b, c, d, e\}$ and edge set $\{ab, bc, cd, be, ce\}$,
- ($\overline{D}_6$, P_5)-free weakly chordal graphs [11], where $\overline{D}_6$ is the complement of the *domino* graph; the domino is a graph with vertex set $\{a, b, c, d, e, f\}$ and edge set $\{ab, bc, cd, de, ef, af, ad\}$,
- ($\overline{P}_6$, P_5)-free weakly chordal graphs [18].

Due to the work of Dahlhaus [9] and He [16], the class of P_4-free graphs can be efficiently recognized in parallel in $O(\log^2 n)$ time with $O(n + m)$ processors; the algorithm in [9] uses the CREW PRAM, whereas the one in [16] uses the CRCW PRAM model. Thus, it is reasonable to ask whether there exist efficient parallel algorithms for recognizing the remaining of the above mentioned subclasses of the P_5-free weakly chordal graphs. Moreover, another interesting question is whether there exist efficient parallel algorithms for finding a perfect order on such graphs (Hayward [13] proposed a sequential $O(n^5)$-time algorithm for finding a perfect order of a P_5-free weakly chordal graph). We leave both questions as open problems.

References

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

[2] S. Arikati and P. Rangan, An efficient algorithm for finding a two-pair, and its applications, *Discrete Applied Math.* **31**, 71–74, 1991.

[3] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis, On the desirability of acyclic database schemes, *J. Assoc. Comput. Machinery* **30**, 479–513, 1983.

[4] A. Berry, J.-P. Bordat, and P. Heggernes, Recognizing weakly triangulated graphs by edge separability, *Nordic J. Computing* **7**, 164–177, 2000.

[5] N. Chandrasekharan, V.S. Lakshmanan, and M. Medidi, Efficient parallel algorithms for finding chordless cycles in graphs, *Parallel Process. Letters* **3**, 165–170, 1993.

[6] K.W. Chong, S.D. Nikolopoulos, and L. Palios, An optimal parallel co-connectivity algorithm, *Theory of Computing Systems* (to appear).

[7] V. Chvátal, Perfectly ordered graphs, *Annals of Discrete Math.* **21**, 63–65, 1984.

[8] V. Chvátal, A class of perfectly orderable graphs, *Report 89-573-OR*, Forschungsinstitut für Diskrete Mathematik, Bonn, 1989.

[9] E. Dahlhaus, Efficient parallel recognition algorithms for cographs and distance hereditary graphs, *Discrete Appl. Math.* **57**, 29–44, 1995.

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

[11] R.B. Hayward, Weakly triangulated graphs, *J. Comb. Theory B* **39**, 200–208, 1985.

[12] R.B. Hayward, Two classes of perfect graphs, *PhD Thesis*, School of Computer Science, McGill University, 1987.

[13] R.B. Hayward, Meyniel weakly triangulated graphs - I: co-perfect orderability, *Discrete Applied Math.* **73**, 199–210, 1997.

[14] R.B. Hayward, C. Hoàng, and F. Maffray, Optimizing weakly triangulated graphs, *Graphs and Combinatorics* **5**, 339–349, 1989.

[15] R.B. Hayward, J. Spinrad, and R. Sritharan, Weakly chordal graph algorithms via handles, *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms (SODA'00)*, 42–49, 2000.

[16] X. He, Parallel algorithm for cograph recognition with applications, *J. Algorithms* **15**, 284–313, 1993.

[17] C.T. Hoàng, On the complexity of recognizing a class of perfectly orderable graphs, *Discrete Applied Math.* **66**, 219–226, 1996.

[18] C.T. Hoàng, F. Maffray, S. Olariu, and M. Preissman, A charming class of perfectly orderable graphs, *Discrete Math.* **102**, 67–74, 1992.

[19] J. Jájá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[20] H. Lerchs, On Cliques and Kernels, *Technical Report*, Department of Computer Science, University of Toronto, 1971.

[21] S.D. Nikolopoulos and L. Palios, Hole and antihole detection in graphs, *Proc. 15th ACM-SIAM Symp. on Discrete Algorithms (SODA'04)* (to appear).

[22] D.J. Rose, Triangulated graphs and the elimination process, *J. Math. Anal. Applications* **32**, 597–609, 1970.

[23] J.P. Spinrad, Finding large holes, *Inform. Process. Letters* **39**, 227–229, 1991.

[24] J.P. Spinrad and R. Sritharan, Algorithms for weakly triangulated graphs, *Discrete Applied Math.* **59**, 181–191, 1995.