INFORMATIQUE THÉORIQUE ET APPLICATIONS

V. B. BALAYOGAN
C. PANDU RANGAN

Parallel algorithms on interval graphs

Informatique théorique et applications, tome 29, n° 6 (1995), p. 451-470.

<http://www.numdam.org/item?id=ITA_1995__29_6_451_0>

© AFCET, 1995, tous droits réservés.

L'accès aux archives de la revue « Informatique théorique et applications » implique l'accord avec les conditions générales d'utilisation (http://www.numdam.org/legal.php). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM

Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques
http://www.numdam.org/

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 29, n° 6, 1995, p. 451 à 470)

PARALLEL ALGORITHMS ON INTERVAL GRAPHS (*)

by V. B. BALAYOGAN ([1]) and C. PANDU RANGAN ([1])

Communicated by Jean BERSTEL

Abstract. – *This paper presents the first optimal, sublogarithmic algorithms for finding the Depth First Search and Breadth First Search trees of an interval graph. Improved/Optimal algorithms are given for Domination problems, and for finding biconnected components, cut vertices and bridges of an interval graph.*

Résumé. – *Cet article présente les premiers algorithmes optimaux, sous logarithmiques pour déterminer les arbres des parcours en profondeur et en largeur de graphes d'intervalles. On donne des algorithmes améliorés ou optimaux pour le problème de domination, et pour déterminer les composantes biconnexes, les points d'articulation et les ponts d'un graphe d'intervalle.*

1. INTRODUCTION

A graph $G = (V, E)$ is said to be an Interval Graph if the vertex set V can be put into an one-to-one correspondance with a set I of intervals on the real line such that two vertices are adjacent in G iff their corresponding intervals intersect. Interval graphs form an important subclass of perfect graphs. Several NP-Complete graph problems admit polynomial solutions when the input is restricted to an interval graph. Hence, extensive studies on the algorithmic and graph theoretic aspects of the interval graphs have been carried out in the past. *See* [G80] [R76] for a detailed discussion on the class of the interval graphs and their applications. In the recent past a growing interest on the design of parallel algorithms has been witnessed. The parallel algorithm design will often call for an entirely new approach for solving the problem at hand as most of the efficient sequential algorithms do not admit "direct parallelisation". Bertossi *et al.* [BB87] were the first ones to present parallel algorithms on interval graphs. They have assumed

(*) Manuscript received March 1992; accepted September 1993, final version received May 1995.

([1]) Department of Computer Science and Engineering, Indian Institute of Technology, Madras, 600 036, India. Email: rangan@iitm.ernet.in

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/95/06/$ 4.00/© AFCET-Gauthier-Villars

that the interval representation is available and solved Maximum weighted clique, Minimum independent set, Maximum clique cover, and Minimum dominating set in $O(\log n)$ time using $O(n^2 \log n)$ processors of a CREW PRAM. Ramalingam and Pandu Rangan [RP87] and Klien [K87] proposed NC algorithms for the interval graph recognition. Recently Moitra *et al.* [MJ88] presented NC algorithms for a variety of problems on interval graphs and proper interval graphs such as depth first search, maximum matching and bandwidth minimisation. NC algorithms for BFS, Hamiltonian paths/cycles, center, shortest paths, minimum colouring, bridges and cut vertices are also reported in the literature. [K89] [OSZ90] [SK91] [SG91] [DC92]. In this paper we present parallel algorithms for the depth first search tree and breadth first tree construction, parallel algorithms for a number of domination problems and algorithms for cut vertices and biconnected components. All our algorithms are either optimum or significant improvement over the previously known results or new and first efficient algorithm for the problem. Comparisons are done in the appropriate sections of this paper.

We use the nearest smallers problem, defined in [BSV88], as a key sub problem and arrive at certain sublogrithmic algorithms for the problems on interval graphs. To the best of our knowledge, these are the first sublogrithmic algorithms on the class of interval graphs. The parallel algorithms presented in this paper are designed for the *Parallel Random Access Machine* (PRAM) model of parallel computation in which all processors have simultaneous, unit time access to a shared memory for reading and writing. Depending on whether reads and writes to a particular memory location are exclusive or concurrent, the PRAMs are further classified into *Exclusive Read Exclusive Write* (EREW), *Concurrent Read Exclusive Write* (CREW) and *Concurrent Read Concurrent Write* (CRCW) models. The CRCW PRAMS used for algorithms in this paper are of the Common Write Type in which multiple processors may attempt to write to the same memory location only if all of them are seeking to write the same value. A parallel algorithm is said to be efficient if it solves the problem in $O((\log n)^k)$ time for some constant k using $POLYNOMIAL(n)$ number of processors, where n is the size of the input. We often call such efficient algorithms as POLY-LOG algorithms or NC algorithms. A parallel algorithm is said to be optimal if the product of processor count and run time is of the same order as the complexity of the best known sequential algorithm or is of the same order as the lower bound to the problem. *See* [R93] [JJ92] for more details on PRAM Models and their algorithmic aspects.

Representation of Input

As is the common convention for parallel algorithms on interval graphs, the algorithms in this paper assume that the right sorted interval representation is available. Some algorithms require the sorted list of all endpoints. Note that given an interval model, Cole's parallel merge sort can be used to obtain either of these input representations in $O(\log n)$ time with $O(n)$ EREW PRAM processors [C86]. However, the adjacency list input imposes a heavy overhead since the best known algorithms for recognition of interval graphs and construction of the interval model take $O(\log^2 n)$ time with $O((n+m)/\log n)$ processors on a CRCW PRAM [K88]. Let us recall an important characterisation of an interval graph in terms of certain ordering or numbering of its vertices. A graph $G = (V, E)$ is an interval graph iff its vertices can be numbered $v_1, v_2, \ldots, v_n$ such that $i < k$ and $(v_i, v_k) \in E$ imply that, for all $i < j < k$, $(v_j, v_k) \in E$ [RP87]. This scheme of numbering the vertices, called the IG-numbering, has been used extensively in the design of several sequential algorithms on Interval graphs. Let I denote the sequence of intervals corresponding to the vertices of the interval graph sorted according to their right endpoints. It is readily seen that the ranks of intervals in I satisfy the conditions for being an IG-numbering. Thus, in what follows, we assume $G = (V, E)$ to be an interval graph given in the form of two arrays and $L = [l_1, l_2, \ldots, l_n]$ and $R = [r_1, r_2, \ldots, r_n]$ where (l_i, r_i) is the interval I_i representing vertex v_i, $1 \leq i \leq n$, and that the R array is sorted in the increasing order.

2. ALGORITHMS

In this section, a *unified* approach is taken to derive optimal parallel algorithms for several interval graph problems. Given the intervals (l_i, r_i) as input, we construct three auxiliary arrays E_D, E_B and E_I, of length $2n$ each. They contain permutations of the l_i and r_i values or their negatives. Three parent functions $P_D(\ )$, $P_B(\ )$ and $P_I(\ )$ are defined based on the corresponding E array by the relation: $P_X(v_i) = v_j$ iff l_j is the right match of r_i when the nearest smallers problem (to be defined later) is solved on E_X, $X \in \{D, B, I\}$. It is then proved that the trees implicity defined by parent functions $P_D(\ )$ and $P_B(\ )$ give the depth first search spanning tree and the breadth first search spanning tree, respectively, of the given interval graph while the path from v_1 to root in $P_I(\ )$ tree is shown to consist of the vertices in the largest independent set of G. The minimum

dominating set, the minimum connected dominating set and the minimum total dominating set of G are similarly shown to be vertex to root paths in trees defined by appropriate parent functions obtained from $P_B(\ )$ and $P_I(\ )$ (by function composition).

The algorithms for cut vertices, bridges and biconnected components work on the segments of the real line demarcated by the endpoints of the intervals.

The following two well known problems are frequently used as key subproblems in our algorithms.

Parallel Prefix Sum

Given an array $A = [a_1, a_2, \ldots, a_n]$ of numbers, the Parallel Prefix Sum problem is to compute the partial sums $S_i = \sum_{1 \leq j \leq i} a_j$, for $1 \leq i \leq n$. This problem can be solved in $O(\log n)$ time using $O(n/\log n)$ processors on an EREW PRAM using standard techniques [R93] [JJ92].

The Nearest Smallers Problem

The input to this problem is an array $A = [a_1, a_2, \ldots, a_n]$ of elements drawn from a totally ordered domain. It is required to find, for each a_i, $1 \leq i \leq n$, the nearest element to its right that is smaller than a_i, if it exists. Such an element, if it exists, is called the *right match* of a_i. This problem can be solved in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM [K89] or in $O(\log \log n)$ time using $O(n/\log \log n)$ processors of a CRCW PRAM [BSV88].

Parallel construction of the DFS tree

Constructing the depth first search tree of an arbitrary graph is one of the problems that is considered "inherently sequential" and there is little likelihood of finding NC algorithms for this problem [R85]. However on restricted classes of graphs such as planar graphs and chordal graphs NC-algorithms have been proposed [H89] [K88]. [MJ88] and [GDSP90] gave $O(n^2/\log n)$ processor $O(\log n)$ time algorithms for constructing the DFS tree of an interval graph when the interval representation is available. Moitra's algorithm used CREW model while GDS algorithm employed EREW model. The processor count was subsequently improved to $O(n)$ in [DC92] for the EREW model. The algorithm presented here reduces the processor count further to $O(n/\log n)$ for EREW PRAM without increasing the time complexity. Clearly our algorithm is optimal and the processor count can

not be reduced any further. Let $T_D(G)$ be the DFS tree of the interval graph G where the traversal started at the vertex n and let $P_D(v_i)$ denote the parent of vertex v_i in the DFS tree. In this section, we develop a method of constructing $T_D(G)$ implicity by finding $P_D(v_i)$ for each vertex v_i. In order to make the construction of the DFS tree deterministic, the convention of choosing the highest numbered vertex in the set of eligible vertices (at each stage of the DFS) is followed. As seen from the Lemma below, this leads to a particularly simple traversal sequence.

LEMMA 1: *The preorder traversal of the vertices of* DFS *tree* $T_D(G)$ *is* $v_n, v_{n-1}, \ldots, v_2, v_1$.

Proof: Easy.

The next lemma gives a new characterisation of the parent function P_D that leads to the algorithm.

LEMMA 2: *The parent of vertex* v_i *in* $T_D(G)$ *is vertex* v_j *iff the following condition holds:* $j = \min\{k \mid k > i,\ I_k$ *intersects* $I_i\}$. *In other words, the parent of any vertex* v *in* $T_D(G)$ *is the lowest numbered vertex among the higher numbered neighbours of* v.

Proof: The proof is by contradiction. Let $S = \{k \mid k > i, I_k intersects I_i\}$ and assume $P(v_i) = v_w$, and not v_j where $j = \min(S)$. $w > i$ since the parent should come earlier in the preorder sequence, according to Lemma 1. This implies that $w \in S$ and since $j = \min(S)$ and $j \neq w$, $j < w$. But $i < j < w$ and I_i intersects I_w imply that I_j intersects I_w. Hence the choice of v_i after v_w contradicts our policy that we choose the largest unvisited neighbour as the next vertex in the DFS traversal. □

The following lemma summaries the properties of the DFS tree in terms of the sorted interval representation.

LEMMA 3: *Let* $T_D(G)$ *be the* DFS *tree of* G *rooted at* n, *obtained on the assumption the higher its* IG-*number of a vertex the higher its priority for being visited next, at every stage of the depth first search. Then, for each vertex* v_i, *the parent of* v_i *in* $T_D(G)$, *denoted* $P_D(v_i)$, *is given by* $P_D(v_i) = v_j$ *where* $j = \min\{k \mid k > i,\ l_k < r_i\}$.

LEMMA 4: *Let* $E_D = [l_1, r_1, l_2, r_2, \ldots, l_n, r_n]$. *Then* $P_D(v_i) = v_j$ *iff* l_j *is the rightmatch of* r_i *when the nearest smallers problem is solved on* E_D [ ].

Proof: $\Leftarrow$: This follows straight form Lemma 2.

$\Rightarrow$: Note that, since the right endpoints form an ascending subsequence in E, no right endpoint can be the rightmatch of any r_i. By lemma 3, $P_D(v_i) = v_j \Rightarrow i \langle j, r_i \rangle l_j$ and for all k, $i < k < j$, $r_i < l_k$. Hence the proof.

Algorithm Parallel-DFS

begin

(1) for $i := 1$ to n in parallel do

$$E[2 * i - 1] := \langle l_i, i \rangle;$$
$$E[2 * i] := \langle r_i, i \rangle;$$

od;

(2) Let the right match of $E(i)$ be denoted by $RM[i]$.

//Note that $E(2i) = \langle r_i, i \rangle$. Thus by Lemma 4, the right match of $E(2i)$, $RM[2i]$, contains the index of parent of v_i.//

(3) Let $RM[2i] = (\alpha_i, \beta_i)$ for $1 \leq i \leq n$.

//β_i is the index and α_i is some end point of the interval.//

$PD(v_i) = v_{\beta_i}$ for $1 \leq i \leq n$.

end.

As noted earlier, solving Nearest-Smallers will take $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM or $O(\log \log n)$ time on $O(n/\log \log n)$ CRCW PRAM processors. Hence,

THEOREM 1: *Given the right sorted interval representation, the Depth First Search spanning tree of an interval graph can be constructed in* $O(\log n)$ *with* $O(n/\log n)$ *processors on an EREW PRAM or in* $O(\log \log n)$ *time with* $O(n/\log \log n)$ *processors on a CRCW PRAM.* □

Parallel construction of the BFS tree

Although there are NC-algorithms for the BFS Tree construction for a general graph, it suffers from "Transitive Closure Bottleneck". Till today, we are not able to overcome the same and the processor count remains same at $O(n^{2.36})$ for an $O(\log n)$ finish time. For the class of the interval graphs, [DC92] demonstrates how to construct the BFS Tree in $O(\log n)$ time using only $O(n)$ processors in EREW PRAM. We further reduce the processor

count to $O(n/\log n)$ and achieve the optimality. We also derive the first sublogrithmic algorithm for this problem based on the same method.

Let $G = (V, E)$ be an interval graph, given in the right sorted interval representation, and let $T_B(G)$ be its breadth first search spanning tree rooted at n. Let $P_B(v_i)$ denote the parent vertex of v_i in $T_B(G)$. Lemma 5 defines $P_B(\ )$ in terms of the interval representation while Lemma 6 reduces the computation of $P_B(\ )$ to the nearest smallers problem.

LEMMA 5: *The parent of vertex v_i is v_j in $T_B(G)$ iff the following condition holds:* $j = \max\{k \mid k > i,\ l_k < r_i\}$.

Proof: Let $d(v)$ denote the depth of vertex v in $T_B(G)$. The Lemma is proved by showing that for any pair of adjacent vertices v_i and v_j, $|d(v_i) - d(v_j)| \leq 1$. Assume $i < j$. Since v_i and v_j intersect, either v_j is the parent of v_i (in which case the depth difference is 1) or neither is an ancestor of the other. In the latter case let v_c be the lowest common ancestor of v_i and v_j and let $A = (v_i = v_{\alpha(1)}, v_{\alpha(2)}, \ldots, v_{\alpha(a)} = v_c)$ and $B = (v_j = v_{\beta(1)}, v_{\beta(2)}, \ldots, v_{\beta(b)} = v_c)$ be the ancestral paths from v_i to v_c and v_j to v_c respectively. Since $i < j$, $a \geq b$. By construction, $r_{\alpha(1)} < r_{\beta(1)}$ and $(v_{\alpha(1)}, v_{\beta(1)}) \in E(G)$ imply $r_{\alpha(2)} < r_{\beta(2)}$ and $(v_{\alpha(2)}, v_{\beta(2)}) \in E(G)$. Hence, by induction, $r_{\alpha(b)} < r_{\beta(b)}$ and $(v_{\alpha(b)}, v_{\beta(b)}) \in E(G)$. This implies that $v_{\alpha(b)}$ is either v_c (in which case $a = b$) or adjacent to v_c (in which case $v_{\alpha(b+1)} = v_c$ or $a = b + 1$. Hence the result. □

LEMMA 6: *Define $E_B = [r_1, r_2, \ldots, r_n, l_n, \ldots, l_2, l_1]$. Then $P_B(v_i) = v_j$ iff l_j is the rightmatch of r_i when the nearest smallers problem is solved on E_B.*

Proof: Similar to Lemma 4.

Hence,

THEOREM 2: *Given the right sorted interval representation, the Breadth First Search spanning tree of an interval graph can be constructed in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM or in $O(\log \log n)$ time with $O(n/\log \log n)$ processors on a CRCW PRAM.* □

Parallel algorithms for bridges, cut vertices and bi-connected components

For general graphs, the best known algorithms for finding bridges, cut vertices and biconnected components take $O(\log^2 n)$ time using

$O(n^2/\log^2 n)$ processors in EREW PRAM model [TSIN84]. Algorithms specific to interval graphs, which runs in $O(\log n)$ time using $O(n^2/\log n)$ processors, were first presented in [GDSP90]. The processor count was reduced to $O(n)$ in [DC92] [SK91]. The approach followed in [GDSP90] is to first construct the DFS spanning tree from the interval model, and then to use the DFS tree and some associated tree functions to identify the bridges, cut vertices and biconnected components. In this section, algorithms that work directly on the interval model to do the same identification are presented. Given the array of sorted endpoints, the algorithms in this section take $O(\log n)$ time on $O(n/\log n)$ EREW PRAM processors. For the rest of this section, let $S[1 \ldots 2 * n]$ be the array that contains the sorted list of endpoints of the interval graph $G = (V, E)$. Let each $S[j]$ be a triplet $\langle l_i, i, 1 \rangle$ or $\langle r_i, i, -1 \rangle$ according to whether it is a left endpoint or right endpoint (where (l_i, r_i) is the interval representing vertex v_i with IG-number i).

Define $L^{-1}[i]$ and $R^{-1}[i]$ to be the ranks of l_i and r_i, respectively, in the sorted ordering of all endpoints. Define $N_R[i]$ ($N_L[i]$) to be the total number of right (left) endpoints that have ranks not greater than i in the sorted ordering of all endpoints. The $2n$ endpoints of the intervals in G (assumed, without loss of generality, to be unique) define $2n - 1$ segments of the real line demarcated on either side by endpoints. In the discussion that follows, each segment is consistently associated to the endpoint immediately to its left. An interval covers a segment if it contains the segment and $W[i]$ is defined to be the number of intervals that cover the segment that begins at the i-th endpoint from the left. From the definitions of $W[i]$, $N_R[i]$ and $N_L[i]$, it follows that for $1 \leq i \leq 2n$, $W[i] = N_L[i] - N_R[i]$.

Cut Vertices

It is easily seen that a cut vertex of the interval graph, as it appears on the real line, will contain at least one segment over which it is the sole covering vertex. Conversely, the unique covering interval of a segment with $W[i] = 1$ is a cut vertex unless the segment is the first or the last one of a connected component. Note that in the latter case $W[i-1] = 0$ or $W[i+1] = 0$. Since for $1 \leq i \leq 2n$, $|W[i] - W[i+1]| = 1$ (from uniqueness of endpoints), this implies that the covering interval of a segment i is a cut vertex iff $W[i-1] = 2$, $W[i] = 1$ and $W[i+1] = 2$. As can easily be seen, the interval I_i covers segments from $L^{-1}[i]$ to $R^{-1}[i] - 1$. Hence, a vertex v_i is a cut vertex iff there exists a j, $L^{-1}[i] \leq j \leq R^{-1}[i] - 1$, such that

$W[j] = 1$ and $W[j-1] = W[j+1] = 2$.

Although the pattern $2-1-2$ is identified easily, checking that for the entire range $L^{-1}[i] \ldots R^{-1}[i] - 1$ for each vertex cannot easily be parallelised. This can be overcome as follows: Define $x[i]$ to be 1 if $W[i] = 1$ and $W[i-1] = W[i+1] = 2$ and 0 otherwise. Let $X[i]$ to be $\sum_{1 \le j \le i} x[j]$. Then v_i is a cut vertex iff $X[R^{-1}[i] - 1] > X[L^{-1}[i]]$. We are now ready for the parallel algorithm.

Algorithm Cut-Vertices

begin

/* Create an array S of records with each record containing three components. The first component is an end point of an interval the second one is the index of the interval for which the first component is an end point and the third component is called the sign component which is 1 or -1 depending on whether the first component is the left or the right end point. */

(1) for $i := 1$ to n in parallel do

$$S[2*i-1] := \langle l_i, i, 1 \rangle;$$
$$S[2*i] := \langle r_i, i, -1 \rangle;$$

od;

(2) Sort $(S[1 \ldots 2*n])$ on the first component

Denote the new $S[i]$ by $S[i] := \langle \text{end}[i], \text{id}[i], \text{sign}[i] \rangle$

(3) for $i := 1$ to $2*n$ in parallel do

$$W[i] := \text{sign}[i];$$

if $\text{sign}[i] = 1$ then

$$L^{-1}[\text{id}[i]] := i$$

else

$$R^{-1}[\text{id}[i]] := i$$

fi

od;

(4) Prefix-Sum $(W[1 \ldots 2*n])$;

(5) for $i := 1$ to $2 * n$ in parallel do

$$X[i] := 0;$$

od;

(6) for $i := 2$ to $2 * n - 2$ in parallel do

if $W[i] = 1$ and $W[i-1] = 2$ and $W[i+1] = 2$ then

$$X[i] := 1;$$

fi

od;

(7) Prefix-Sum $(X[1 \ldots 2 * n])$;

(8) for $i := 1$ to $2 * n$ in parallel do

if $X[R^{-1}[i] - 1] > X[L^{-1}[i]]$ then

Output (v_i);

fi

od;

end.

The calls to Prefix-Sum take $O(\log n)$ time on $O(n/\log n)$ EREW PRAM processors. All steps after (2) can be done within the same bounds. Hence,

THEOREM 3: *Given the sorted endpoint representation of an interval graph G, all cut vertices of G can be identified in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM.* □

Bridges

Let $e = (v_i, v_j)$, $i < j$, be a bridge of the given interval graph. Since deletion of e disconnects the graph, no other interval covers the segment of overlap $[l_j, r_i]$ on the real line. Hence, if $k = L^{-1}[j]$, $W[k] = 2$ and $W[k-1] = 1$. If $W[k+1] > 2$, $k+1$ cannot be a right endpoint and e is not a bridge. Hence, the condition that $W[k] = 2$, $W[k-1] = W[k+1] = 1$ characterise a bridge. To identify v_i and v_j, note that $S[k]$ is the left endpoint of v_j and $S[k+1]$ is the right endpoint of vi. The complete parallel algorithm is given below:

Algorithm Bridges

begin
Perform Steps 1 and 2 as in the previous algorithm **cut-vertices**
(3) for $i := 1$ to $2 * n$ in parallel do

$$W[i] := \text{sign}[i];$$

od;
(4) Prefix-Sum $(W[1 \ldots 2 * n])$;
(5) for $i := 2$ to $2 * n - 2$ in parallel do
if $W[i] = 2$ and $W[i-1] = 1$ and $W[i+1] = 1$ then
Output $((v_{\text{id}[i]}, v_{\text{id}[i+1]}))$;
fi
od;
end.

The calls to Prefix-Sum take $O(\log n)$ time on $O(n/\log n)$ EREW PRAM processors. All other steps after (2) can be done within the same bounds. Hence,

THEOREM 4: *Given the sorted endpoint representation of an interval graph G, the bridges in G can be identified in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM.* □

Biconnected Components

Observe that biconnected components and cut vertices of proper interval graphs form a linear chain-and-link structure. Although this is not true for general interval graphs, a direct characterisation in terms of the segments on the real line is still possible. A segment i with $W[i] = 1$ will have two different biconnected components on either side of it. Hence for a connected interval graph, counting the segments with $W[i] = 1$, $1 \leq i \leq 2n - 2$, assigns a "left to right" numbering to all the biconnected components of G. Consider the problem of identifying the biconnected component that contains the edge (v_i, v_j). The segments between $\max(L^{-1}[i], L^{-1}[j])$ and $\min(R^{-1}[i] - 1, R^{-1}[j] - 1)$ lie on the same biconnected component and any segment in this range can be chosen as the representative to find the rank (from left) of the biconnected component that contains (v_i, v_j). However for if the graph contains more than one connected component, a segment i with

$W[i] = 1$ and $W[i+1] = 0$ does not begin any new biconnected component and it should not be counted (the new component begins at $W[i+2] = 1$). This can now be implemented as a parallel preprocessing algorithm that answers queries in $O(1)$ time.

Algorithm Biconnected-Components

begin
Perform Steps 1 and 2 as in the previous algorithm **cut-vertices**
(3) for $i := 1$ to $2 * n$ in parallel do

$$W[i] := \text{sign}[i];$$

if $\text{sign}[i] = 1$ then

$$L^{-1}[\text{id}[i]] := i$$

fi
od;
(4) Prefix-Sum $(W[1 \ldots 2 * n])$;
(5) for $i := 1$ to $2 * n - 1$ in parallel do
(5.1) if $W[i] = 1$ and $W[i+1] = 2$ then

$$BiCC[i] := 1$$

else

$$BiCC[i] := 0$$

fi
od;
(6) Prefix-Sum $(BiCC[1 \ldots 2 * n - 1]4)$
end.

Algorithm Find_Owner_BiCC $((v_i, v_j))$

begin
(1) Output $(\text{BiCC}[\max[L^{-1}[i], L^{-1}[j]]])$
end.

The complexity analysis of the algorithms are straightforward.

Hence,

THEOREM 5: *Given the sorted endpoint representation of an interval graph G, the biconnected components can be identified in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM such that edge membership queries on the biconnected components can be answered in constant time.* □

Some Other Problems Solvable by Overlap Counting

Given the sorted endpoint representation of an interval graph, several other problems also can be efficiently solved in parallel by the technique of counting overlapping intervals of segments. An algorithm to identify the *k-connected components* of a $k-1$ connected interval graph can easily be obtained by generalising algorithm for biconnected components as follows: Change step (5.1) to

If $W[i] = k-1$ and $W[i+1] = k$.

Several clique problems too can be solved using this technique. Note that a segment i with $W[i] > \max(W[i-1], W[i+1])$ corresponds to a maximal clique of the graph. Hence the problems of finding *the maximum clique, weighted maximum clique, maximum weighted maximal clique* etc. can all be solved in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM.

REMARK 1: *Given a sorted end point representation of an interval graph, all the following problems can be solved in $O(\log n)$ time using $O(n/\log n)$ processors in EREW PRAM model.*

a) k-connected components

b) (Weighted) maximum clique.

Parallel algorithms for dominating sets

Finding a dominating set is a typical example of a problem that is intractable in general graphs but admits an efficient solution in some special classes of graphs. Interval graphs are among the classes of graphs for which there are efficient algorithms for sequentially finding the minimum set for domination, connected domination, total domination and independent domination. A unified approach to the weighted case of the sequential problem is given in [RP88]. The first parallel algorithm for minimum dominating set problem on Interval graph was presented in [BB87]. It

takes $O(\log n)$ time but employs $O(n^2/\log n)$ processors of a CREW PRAM. The processor count was subsequently improved to $O(n)$ in the weaker EREW PRAM model in [K89] and [OSZ90]. In this section, a unified approach for obtaining the independent set and various dominating sets in parallel is presented. All the algorithms run in $O(\log n)$ time but employ only $O(n/\log n)$ processors of EREW PRAM.

First, the nearest smallers problem is used to evaluate two parent functions, $P_B(\ )$ and $P_I(\ )$ (defined below). Then corresponding to each dominating set, a new parent function is defined in terms of $P_B(\ )$ and $P_I(\ )$. The domination problems are then reduced to finding an ancestral path from a specific vertex to the root in the trees defined by the respective parent functions. The problem of identifying the vertices of a path of a tree in parallel is solved by the Eulerian tour techniques.

The Maximum Independent Set

Define $P_I(v_i) = v_j$ iff $j = \min\{k \mid l_k > r_i\}$ and define $T_I(G)$ to be the tree formed by the parent function $P_I(\ )$ defined above. Let I be the vertices in the path $v_1, P_I(v_1), P_I(P_I(v_1)), \ldots$. By induction on the IG-number of vertices in I it can easily be shown that I is an independent set and that $|I|$ is at least as large as the size of any other independent set.

LEMMA 6: *Let* $E_I = [-r_n, -r_{n-1}, \ldots, -r_1, -l_1, -l_2, \ldots, -l_n]$. *Then* $P_I(v_i) = v_j$ *iff the rightmatch of -ri is -lj when the nearest smallers problem is solved on EI.*

Proof: Easy, details are omitted. □

The next lemma proves an important result that is necessary for efficient extraction of the independent set from the tree TI(G) using Eulerian tour.

LEMMA 7: *Let* $P_B(\ )$ *and* $P_I(\ )$ *be the parent functions of the BFS and independence trees. Then, for* $i < j$,

(i) $P_B(v_i) = P_B(v_j) = v_p \Rightarrow$ for all k, $i < k < j$, $P_B(v_k) = v_p$

(ii) $P_I(v_i) = P_I(v_j) = v_q \Rightarrow$ for all k, $i < k < j$, $P_I(v_k) = v_q$

Proof: Straightforward from the property of IG-numbering. □

Any function that satisfies the above lemma is said to have the adjacent children property. By Lemma 7, all children of a vertex (in either tree) are numbered consecutively. We now give an algorithm to compute the path from a leaf to the root in a tree given that (i) The tree is represented in

the form of a parent (P) array and (ii) The P array satisfies the adjacent children property.

Assume that v_1 is the given leaf.

A) Find the adjacency list

A.1) Assign $\log n$ elements of P to each of $n/\log n$ processors.

A.2) Each processor scans the $\log n$ elements assigned to it from left to right, and if $P[i] \neq P[i+1]$, then it sets $\mathrm{Max}[P[i] = i$, $\mathrm{Min}[P[i+1]] = i+1$, where Min and Max are the first and last children of a node.

A.3) $\mathrm{Min}[i] \ldots \mathrm{Max}[i]$ gives the children of node i for $i \leq i \leq n$.

B) From this adjacency information, we construct a directed graph by replacing each edge in the tree by a forward and a backward edge. Then, we construct the Eulerian tour using $n/\log n$ processors and $\log n$ time [$KR88$].

C) Find the preorder number of each node.

D) Find the last occurence of v_1 and then find the first occurence of the root after the last occurence of v_1. The position of the last occurence of v_1 is found out by the following method: Assign a weight of 1 to an edge in the tour if it is incident on v_1 and 0 otherwise. Perform a parallel prefix sum. The values got are in nondecreasing order. The value associated with the last edge of the tour gives the number of times v_1 occures in this tour. The edge where this value first occures, which can be found out by a binary search, is the last occurence of v_1. In a similar manner, the position of the root can be found out.

E) Take all the vertices in the tour that exist between the above computed positions in the Eulerian tour. Remove those vertices whose preorder numbers are greater than the preorder number of v_1. The list of preorder numbers got will be in nonincreasing order. Removing duplicates from this list will give the vertices in the path from v_1 to root. Correctness of the above algorithm is established as follows: Consider the portion of the Eulerian tour computed. This is a walk from the vertex v_1 to the root. This walk, when condensed by removing duplicating vertices, will give a path from v_1 to the root, this should be identical to the path computed. Thus, the walk will contain all the required vertices. We can identify the unnecessary vertices by the fact that their preorder number will be greater than that of v_1. This is because, vertices which have preorder number less than v_1 and do not belong to the path from v_1 to the root, will have a preorder number less than of v_1. Hence the correctness follows.

Complexity: Step A requires $O(\log n)$ time with $n/\log n$ processors. Step B can be done in $O(1)$ time using n processors, and hence in $O(\log n)$ time with $O(n/\log n)$ processors. Steps C, D use parallel prefix algorithm, and hence can be done optimally. Step E involves deletion of marked elements from a list, which requires $O(\log n)$ time with $O(n/\log n)$ processors.

THEOREM: *Given a P array representation of a tree which satisfies the adjacent children property, the path from a leaf to the root can be computed in $O(\log n)$ time using $n/\log n$ processors on an EREW-PRAM.*

LEMMA 8: *Given the right sorted interval representation, a maximum independent set of an interval graph can be constructed in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM.* □

COROLLARY: *A* **minimum clique cover** *of an interval graph can be constructed in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM.*

Proof: Let $I = \{v_{\alpha(1)}, \ldots, v_{\alpha(k)}\}$ be the maximum independent set as obtained above and let $r_{\alpha(k+1)} = \infty$. For $1 \leq i \leq k$, define $C_i = \{v_j \mid r_{\alpha(i)} \leq r_j < r_{\alpha(i+1)}\}$ and $C_I = \{C_i \mid 1 \leq i \leq k\}$. Note that C_I is a partition of V and each C_i induces a clique. C_I is a minimum clique cover since no clique cover can have a size smaller than $|I|$. C_I can be constructed from I by parallel prefix in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM. □

REMARK 2: *For the rest of the section on dominating sets, the details common to connected dominating sets, total dominating sets and the dominating sets are omitted and only the correctness of the composite parent functions and the proof of adjacent children property are given.*

Hence time complexity of $O(\log n)$ with $O(n/\log n)$ EREW PRAM processors applies to all algorithms below.

Minimum Dominating Set

LEMMA 9: *Let $D = \{v_{\alpha(1)}, v_{\alpha(2)}, \ldots, v_{\alpha(k)}\}$ where the sequence of $\alpha(\ )$ values are defined by: $v_\alpha(1) = P_B(v_1)$ and, for $1 < i \leq k$, $v_{\alpha(i)} = P_B(P_I(v_{\alpha(i-1)}))$. Then D is a minimum dominating set of G.*

Proof: Easy.

LEMMA 10: *The parent function* $P_B(P_I(\ ))$ *satisfies the adjacent children property.*

Proof: If the rightmost and leftmost children, say v_p and v_q, of $P_B(P_I(v))$ have the same $P_I(\ )$ value then the adjacent children property of $P_I(\ )$ gives the required result. Else, by the property of IG-numbering every vertex between v_p and v_q has a $P_I(\ )$ value bounded by those of v_p and v_q and the result follows from the adjacent children property of $P_B(\ )$. □

THEOREM: *Given a sorted interval representation, the minimum dominating set problem can be solved for Interval graphs in* $O(\log n)$ *time using* $O(n/\log n)$ *processors of an EREW PRAM.*

Proof: Follows from Lemma 9, Lemma 10, and Remark 2.

Minimum Connected Dominating Set

LEMMA 11: *Let* $D_C = \{v_{\alpha(1)}, v_{\alpha(2)}, \ldots, v_{\alpha(k)}\}$ *where the sequence of* $\alpha(\ )$ *values are defined by:* $v_{\alpha(1)} = P_B(v_1)$ *and for* $1 < i \leq k$, $v_{\alpha(i)} = P_B(P_B(v_{\alpha(i-1)}))$. *Then* D_C *is a minimum connected dominating set of* G.

Proof: In any minimum connected dominating set D'_C of G, there exists a vertex v_c which is adjacent to v_1. The result easily follows from induction on i on the subgraph induced by the first $\alpha(i)$ vertices in the IG-numbering. As an alternative proof, one can use the fact that the above sequence is actually the v_1 to root shortest path in the BFS spanning tree of G. □

LEMMA 12: *The parent function* $P_B(P_B(\ ))$ satisfies the adjacent children property.

Proof: If the rightmost and leftmost children, say v_p and v_q, of $P_B(P_B(v))$ have the same $P_B(\ )$ value then the adjacent children property of $P_B(\ )$ gives the required result. Else, by the property of IG-numbering every vertex between v_p and v_q has a $P_B(\ )$ value bounded by those of vp and vq and the result follows from the adjacent children property of $P_B(\ )$. □

THEOREM: *Given a sorted interval representation, the minimum connected dominating set problem can be solved for Interval graphs in* $O(\log n)$ *time using* $O(n/\log n)$ *processors of an EREW PRAM.*

Proof: Follows from Lemma 11, Lemma 12, and Remark 2.

Minimum Total Dominating Set

LEMMA 13: *Let* $D_T = \{ v_{\alpha(1)}, v_{\alpha(2)}, \ldots, v_{\alpha(k)} \}$ *where the sequence of* $\alpha(\ )$ *values are defined by the recurrence:* $v_{\alpha(1)} = P_B(v_1)\ v_{\alpha(2)} = P_B(v_{\alpha(1)})$ *and for* $1 < i \leq k/2$, $v_{\alpha(2i-1)} = P_B(P_I(v_{\alpha(2i-2)}))\ v_{\alpha(2i)} = P_B(P_B(P_I(v_{\alpha(2i-2)})))$. *Then DT is a minimum total dominating set of G.*

Proof: Define $\beta(\ )$ such that, $v_{\beta(i)} = P_B(v_{\alpha(2i)})$. Let D'_T be an arbitrary minimum total dominating set of G. Then there are two adjacent vertices in D'_T, say v_d and v_{dd}, at least one of which is adjacent to v_1. Since $v_{\alpha(1)}$ is the right most vertex adjacent to v_1 and $v_{\alpha(2)}$ is the right most vertex adjacent to $v_{\alpha(1)}$, it follows that $\{ v_d, vdd \}$ cannot dominate a larger set than $\{ v_1, v_2, \ldots, v_{\beta(1)} \}$. From the properties of the IG-numbering, it is easily seen that no minimum dominating set (of any type) of the subgraph induced by the vertex set $\{ v_{\beta(1)+1}, \ldots, v_n \}$ need contain a vertex in $v_1, \ldots, v_{\beta(1)}$. The result follows from this by induction. □

The composite parent function defined for D_T is not a simple recursive relation as was the case with other dominating sets. However by taking every odd numbered vertex in D_T, is possible to get the single parent function $P_B(P_I(P_B(\ )))$. For every vertex v in the set obtained from this parent function, $P_B(v)$ should also be added. (If v_n is in the first set, add an arbitrary neighbour of v_n.)

LEMMA 14: *The parent function* $P_B(P_I(P_B(\ )))$ *satisfies the adjacent children property.*

Proof: As in the earlier cases, this can be proved from the property of IG-numbering and the adjacent children property of the individual functions. □

THEOREM: *Given a sorted interval representation, the minimum total dominating set problem can be solved for Interval graphs in* $O(\log n)$ *time using* $O(n/\log n)$ *processors of an EREW PRAM.*

Proof: Follows from Lemma 13, Lemma 14, and Remark 2.

3. CONCLUSION

We have presented the first sublogarithmic and optimal algorithms for finding the DFS tree and BFS tree. We have also improved the complexity for domination problems and our algorithms are all derived in a unified

manner. We have used the nearest smallers and the IG ordering to improve the complexity of the algorithms. It remains to see if such techniques can be applied to wider classes of graphs such as chordal graphs and cocomparability graphs.

REFERENCES

[BSV88] O. Berkman, B. Schieber and U. Vishkin, *Some doubly logarithmic optimal algorithms based on nearest smallers*, Research Report RC 14128 (#63291), IBM Research Division, Israel, 1988.

[BB87] A. A. Bertossi, and M. A. Bonuccelli, Some parallel algorithms on interval graphs, *Discrete Applied Mathematics*, 1987, *16*, pp. 101-111.

[C86] R. Cole, Parallel merge sort, *Proc. 27th Annual Symposium on the Foundations of Computer Science*, 1986, pp. 511-516.

[GDSP90] G. D. S. Ramkumar and C. Pandu, Rangan, Parallel algorithms on interval graphs, *Volume 3 in the Proc. 1990 International Conference on Parallel Processing*, 1990, pp. 72-75.

[G80] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, USA, 1980.

[K88] P. Klein, *Efficient parallel algorithms on chordal graphs*, Laboratory for Computer Science, MIT, USA, 1988. Also Appeared as Chapter 8 in [R93].

[MJ88] A. Moitra and R. Johnson, *Parallel algorithms for maximum matching and other problems on interval graphs*, TR 88-927, Cornell University, Ithaca, USA, 1988.

[RP88] G. Ramalingam and C. Pandu, Rangan, A unified approach to domination problems on interval graphs, *Information Processing Letters*, 1988, *27*, pp. 271-274.

[R85] J. H. Reif, Depth First Search is inherently sequential, *Information Processing Letters*, 1985, *20*, pp. 229-234.

[R93] J. H. Reif, *Synthesis of Parallel Algorithms*, Morgan Kaufmann, California, USA, 1993.

[R76] F. S. Roberts, *Discrete Mathematical Models with Applications to Social, Biological and Environmental problems*, Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1976.

[SW88] J. E. Savage and M. G. Wloka, *A parallel algorithm for channel routing, Proceedings of WG'88, Graph-theoretic Concepts in Computer Science* (published as Lecture Notes in Computer Science, Springer-Verlag, New York, 1988).

[TC84] Y. H. TSIN and F. Y. CHIN, Efficient parallel algorithms for a class of graph theoretic problems, *SIAM Journal of Computing*, 1984, *13*, pp. 580-599.

[SG91] M. A. SRIDHAR and S. GOYAL, Efficient parallel Computation of Hamiltonian Paths and Circuits in Interval Graphs, *Proc. Int. Conf. On Parallel Processing*, Vol. 3, 1991, pp. 83-90.

[K89] S. K. KIM, Optimal Parallel Algorithms on Sorted Intervals, *Proc. 27th Annual Allerton Conf. on Comm., control and Computing*, 1989, pp. 766-775.

[OSZ90] S. OLARIU, J. L. SCHWING and J. ZHANG, Optimal Parallel Algorithms for Problems Modelled by a Family of Intervals, Proc. 28th Annual Allerton Conf. on Comm., *Control and Computing*, 1990, pp. 282-291.

[SK91] Alan P. SPRAGUE and K. H. KULKARNI, *Optimal Parallel algorithms for finding the Cut vertices and Bridges of Interval graphs*, Technical report, University of Alabama, USA, June, 1991.

[DC92] S. K. DAS and C. C. Y. CHEN, *Efficient Parallel Algorithms on Interval graphs*, Technical report, Department of Computer science, University of North texas, USA, 1992.

[JJ92] Joseph JA JA, *An Introduction to Parallel Algorithms*, Addison Wesley, USA, 1992.