Parallel Maximal Cliques Algorithms for Interval Graphs with Applications

Chi Su Wang and Ruay Shiung Chang
Department of Information Management
National Taiwan Institute of Technology
Taipei, Taiwan, Republic of China

Abstract

In this paper, an O(n *log* n) time algorithm for finding all the maximal cliques of an interval graph is proposed. This algorithm can also be implemented in parallel in O(*log* n) time using O(n^2) processors. The maximal cliques of an interval graph contain important structural information. Many problems on interval graphs can be solved after all the maximal cliques are known. It is shown that cut vertices, bridges, and vertex connectivities can all be determined easily after the maximal cliques are known. Finally, the all-pair shortest path problem for interval graphs is solved based on the relationship between maximal cliques. The all-pair shortest path algorithm can also be parallelized in O(*log* n) time using O(n^2) processors.

1. Introduction

An undirected graph $G=(V, E)$ is called an interval graph if there exists a set of intervals S of a linearly ordered set (like the real line) such that each vertex in V can be put into one-to-one correspondence with S and two vertices are connected by an edge of G if and only if their corresponding intervals have nonempty intersection. S is the interval model of G. Interval graph is a proper subset of chordal graph and is among the most useful mathematical structures for modeling real world problems[4]. Many NP-complete problems can be solved in linear or polynomial time on interval graphs, such as hamiltonian circuit[8, 9], chromatic number[13], domatic number[1, 15], and minimum bandwidth[6]. Given a graph $G=(V,E)$, a subset $C \subseteq V$ is called a *clique* if the elements of C form a complete subgraph of G. C is called *maximal* if there is no clique of G which properly contains C as a subset. Finding a maximal clique of any given graph is easy. But, generating all the maximal cliques of a graph requires some work. For interval graphs, this is especially important and useful since many solutions on problems about interval graphs include maximal cliques finding as a first step[13, 15]. Usually, they refer to the method in [4], which requires O($n+m$) time, where n is the number of vertices and m is the number of edges. But, this method is usually used to find all the maximal cliques of a chordal graph. (Since interval graphs are a proper subset of chordal graphs, it also works for interval graph.) One wonders whether a simpler algorithm exists for interval graphs.

0-8186-6507/94 $4.00 © 1994 IEEE

Indeed, it is. In the next section, based on an observation about maximal cliques on interval graphs, an $O(n \log n)$ time algorithm for finding all the maximal cliques of an interval graph is given. ($O(n)$ if endpoints are presorted.) Parallel implementation takes $O(\log n)$ time using $O(n^2)$ processors. A byproduct in Section 2 is if one knows all the maximal cliques of an interval graph, cut vertices, bridges, and vertex connectivities can also be determined easily. In Section 3, the all-pair shortest path problem on interval graphs is solved in a more staightforward and simpler way than that in [7]. Furthermore, it can be parallelized easily in $O(\log n)$ time using $O(n^2)$ processors. Finally, conclusions and possible future researches are discussed in Section 4.

2. Algorithms for Maximal Cliques, Cut vertices, Bridges, and Vertex Connectivities

In this section, we devise an elegant algorithm for finding all the maximal cliques of an interval graph based on an interesting observation. Once the maximal cliques are known, cut vertices, bridges, and vertex connectivities can then be determined easily. Without loss of generality, assume open interval model for interval graphs, no two intervals have a common endpoint and intervals are labelled according to ascending left endpoints.

Theorem 1. Given an interval graph with n intervals, denote the sorted endpoints sequence by $d1, d2, \ldots, d2n$ with $di < dj$ for $1 \le i < j \le 2n$. Then all the intervals that cover on open interval $(di, di+1)$ form a maximal clique if and only if di is a left endpoint and $di+1$ a right endpoint, not necessarily of the same interval.

Proof. Omitted. □

From the above theorem, if S is a maxiaml clique of an interval graph, it must include a common interval (a, b), where a is a left endpoint and b is a right endpoint. Assume x is the interval with left endpoint a and y is the interval with right endpoint b. Define x as the *rightmost interval* of S and S as the *rightmost maxiaml clique* that contains y. For example, in Fig. 2., the maximal clique containing interval $(a5, b3)$ is $C2$. The rightmost interval of $C2$ is 5 and the rightmost maximal clique containing 3 is $C2$.

Based on Theorem 1, we propose a new way to find the maximal cliques of the interval graph G provided the interval model is given. Before formally listing the algorithm, we first describe the concept of the method. First, sort the $2n$ endpoints in ascending order and initialize the maximal clique set to be empty. Then scan the sorted list. If a left endpoint is met, inserting the interval with the current left endpoint into the maximal clique set. If a right endpoint is met and the previous scanned endpoint is a left endpoint, report all the intervals in the maximal clique set as a new maximal clique and delete the interval with the current right endpoint from the maximal clique set. If the previous scanned endpoint is also a right endpoint, just delete the interval with the current right endpoint from the maximal clique set. Take Fig. 2. as an example. Assume ai is the left endpoint of interval i and bi is the right endpoint. First, let the maximal clique be empty, scan the endpoints from $a1$, then $a2$, $a3$, and $a4$. Intervals $1, 2, 3$ and 4 are put into the maximal clique set. Next, $b1$ is scanned. Since the previous endpoint scanned is $a4$, we report $C1 = \{1, 2, 3, 4\}$ and delete interval 1 from the maximal clique set. $a5$ is scanned. Insert interval 5. $b3$ is

scanned. Report $C2 = \{2, 3, 4, 5\}$ and delete interval *3*. *b2* is scanned, a consecutive right endpoint. Delete interval *2*. Continuing, we will have the *4* maximal cliques as in Fig. 2.

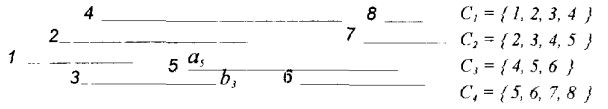

Fig. 2. An interval graph and its maximal cliques.

The above method can be implemented in O($n \log n$) time sequentially. In the following, we show that it can be parallelized.

Algorithm 1. *Find Maximal Cliques in Parallel*

Input: an interval model F.

Output: all the maximal cliques of F.

1. Sort the $2n$ endpoints in ascending order.

2. For each left endpoint, use parallel prefix to find all intervals containing this left endpoint.

3. For each left endpoint, say *ax*, queries its next endpoint to its right. If the next endpoint is a right endpoint, reporting the intervals containing *ax*. (These intervals form a maximal clique and *ax* is the rightmost interval of this maximal clique.)

We describe Steps 2 and 3 of Algorithm 1 in detail. In Step 2, the use of parallel prefix is to record intervals that contain a left endpoint. Now, we describe how to do this. Assume *d1*, *d2*, ..., *d2n* are the $2n$ endpoints sorted in ascending order. For each *di*, $1 \le i \le 2n$, if *di* is the left endpoint of an interval *j*, assign a value of $+j$ to *di*. If *di* is the right endpoint of an interval *j*, assign a value of $-j$ to *di*. Next, use parallel prefix to compute $\sum_{i=1}^{k} d_i$, for $1 \le k \le 2n$. But instead of performing real addition operation, we want to compute how many positive entries are in $\sum_{i=1}^{k} d_i$ and how many corresponding negative entries are there. Note that it is not possible to have a negative entry in $\sum_{i=1}^{k} d_i$ without a corresponding positive entry since the left endpoint is encountered first. For this purpose, assign n processors, numbered as *P1*, *P2*, ..., *Pn*, to each endpoint. The initial states of these n processors on each endpoint are determined as follows. For each interval i, $1 \le i \le n$, *Pi* is set to *on* whether it is on the left or right endpoint and the states of all other processors are *off*. When computing $\sum_{i=1}^{k} d_i$ for $1 \le k \le 2n$, the summation operation corresponds to the n processors passing their states through shared memories to another set of n processors. The state changing rules are described below. Two *on*'s result in *off*, *on* plus *off* is *on*, *off* plus *off* is *off*, and *off* plus *on* is *on*. After the parallel prefix, the processors that are *on* for each left endpoint indicate the intervals that pass through this endpoint plus the interval with this left endpoint. For example, in Fig. 3. at *a4*, after the

parallel prefix, *P1*, *P2*, *P3*, and *P4* should be *on*. Step 3 is the application of Theorem *1*. From Theorem *1*, we have the maximal clique occurs in the open interval (a left endpoint, a right endpoint). Thus, in Step 3, each left endpoint queries whether its successive endpoint to the right is a right endpoint. While an affirmative answer is obtained, a maximal clique is found by reporting the intervals which contain this left endpoint. (The processors which are *on* report.) In Fig. 3, let $+i$ and $-i$ denote the left and right endpoints of interval i, respectively. After parallel prefix, the results are shown in the bottom part of this figure.

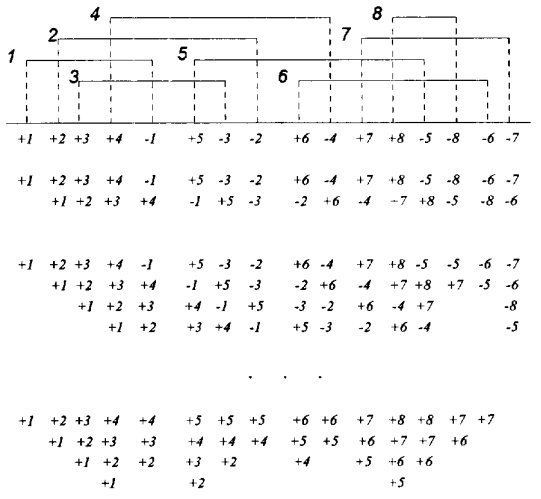

Fig. 3. Find maximal cliques in parallel

Theorem 2. Given an interval model F, Algorithm 1 can find all the maximal cliques of F in parallel with O($log\ n$) time and O(n^2) processors.

Proof. The correctness is obvious from Theorem 1 and the above discussion. Next, we analyze the complexity. Since each endpoint is assigned n processors, it needs O(n^2) processors. Step 1 needs O($log\ n$) time[2, 10, 17]. Parallel prefix in step 2 needs O($log\ n$) time[3]. As for step 3, the operation of query needs constant time and the report needs constant time. Consequently, Algorithm 3 needs O(log n) time with O(n^2) processors. This completes the proof. □

Many problems on interval graphs can be solved easily after all the maximal cliques are found. In the following, we apply it to find the cut vertices, bridges, and vertex connectivities. A vertex is called a cut vertex if its removal results in G having more components than G has. An edge of G is called a bridge if its removal has the same effect as removing a cut vertex. The vertex connectivities of a graph G is defined to be the

minimum number of vertices whose removal disconnects G. Sprague and Kulkarni[16] use *density sequence* (The density of an endpoint is the number of intervals which contain this endpoint.) to solve the first two problems for interval graphs in parallel. It needs $O(\log n)$ time using n processors in EREW PRAM model if the endpoints are not presorted. If the endpoints are presorted, it needs $O(\log n)$ time using $(n / \log n)$ processors. [5] solves the vertex connectivities in $O(\log n)$ time using $O(n)$ processors for unsorted case or in $O(\log n)$ time using $O(n / \log n)$ processors for sorted case. But, [5] just calculates the size of vertex connectivities. In this section, we can also find the k-vertex cutset and k vertex-disjoint paths for a k-connected interval graphs from the maximal cliques. First, we note a useful property about maximal cliques of an interval graph.

Theorem 3[4]. A graph G is an interval graph if and only if the maximal cliques can be linearly ordered such that the maximal cliques containing a vertex v occur consecutively.

Theorem 4. Let $C1, \ldots, Cr$ be the maximal cliques of an interval graph G. (1) G has cut vertex if and only if $Ci \cap Ci+1 = \{x\}$ for some i, $1 \le i \le r-1$, that is $Ci \cap Ci+1$ has only one element x. (2) G has brdige if and only if some maximal clique Ci contains only two vertices.

Proof. Omitted. □

We can extend the result about cut vertices one step further.

Theorem 5. Let $C1, C2, \ldots, Cr$ be the maximal cliques of interval graph G, and $Si = Ci \cap Ci+1$ for $1 \le i \le r-1$, and $k = \min |Si|$, then the *vertex connectivity* of G is k.

Proof. Omitted. □

From Theorems 4 and 5, parallel algorithms for cut vertices, bridges, and vertices connectivities can all be easily devised.

3. All-Pair Shortest Path

In this section, we discuss the problem of all-pair shortest path. Madhav et al.[7] use a special data structure called *neighborhood tree* to solve this problem. Here, we propose a new approach for this problem by using maximal cliques. This is more straightforward and simpler than [7]. First some notations: ① *rightmost-clique*(j) denotes the rightmost maximal containing j, and *rightmost-interval*(i) denotes the rigthmost interval (the interval with the largest index) in the ith maximal clique. ② *clique-link*(i) is the maximum of *rightmost-clique*(j) for any interval j in the ith maximal clique (= $max\{$*rightmost-clique*(j)$\}$, $j \in Ci$), and *clique-link-interval*(i) is the interval with *rightmost-clique* value equals to *clique-link*(i) in the ith maximal clique. (Hereafter, j denotes an interval, i denotes the index of the maximal clique, and the *clique-link* value is an index of some maximal clique.) ③ $D(x, y) = (d, v)$ means the distance between x and y is d and v is the "*via*" interval. The "*via*" interval will be explained in detail later. If "*via*" interval is 0, (x, y) is an edge and $d = 1$ or $x = y$. Before proceeding, we use an example to illustrate the idea. For interval 1 in Fig. 4, we find the rightmost interval which interval 1 can arrive within one step, i.e., interval 4. Second, we find an interval from the neighborhood of interval 1, *i.e.*, from $C1$, with the maximum *rightmost-clique* value. This interval is interval 4 with the maximum *rightmost-clique* value 3(*clique-link*(1) = 3 and

clique-link-interval(*1*) = *4*). This means interval *1* can arrive intervals *5* and *6* via interval *4*, and interval *4* is called the "*via*" point. Next, we find the interval with the maximum *rightmost-clique* value in *C3*, this interval is interval *5* with the maximum *rightmost-clique* value *4*(*clique-link*(*3*) = *4* and *clique-link-interval*(*3*) = *5*). Thus, interval *4* can arrive intervals *7* and *8* via interval *5*, and interval *5* is also a "*via*" point. This indicates interval *1* can arrive interval *8* via intervals *4* and *5* and the distance of intervals *1* and *8* is *3*. The other distances can also be computed by the same way. In Fig. 4, only some key entries are listed. Others are the same as their first nonblank right entry. For example, if we query $D(1, 7)$, we have $D(1, 7) = (3, 5)$ since it first nonblank right entry is $D(1, 8)$.

The sketch of the parallel algorithm is as follows.

Algorithm 2. *Find Distance in Parallel*

1. Find maximal cliques and *rightmost-clique* in parallel.
2. Find *clique-link* and *clique-link-interval* in parallel.
3. Find *rightmost interval string* for each maximal clique.
4. Compute distance in parallel.

Before proving the correctness of Algorithm 2, we need the following Theorem.

Theorem 7[13]. A graph $G=(V, E)$ is an interval graph if and only if there exists a linear order $<$ on V such that for every choice of vertices u, v, w with $u < v < w$, $(u, w) \in E$ implies $(u, v) \in E$.

The linear order in the above Theorem can be achieved by arranging the intervals in ascending order with respect to left endpoints.

Theorem 8. Algorithm 2 correctly computes the all-pair shortest path for an interval graph in parallel.

Proof. Let $C1, \ldots, Cr$ be the maximal cliques of the input graph. From Theorem 7, we know that if $u < v < w$ and u and w have edge, u and v also have edge. Hence, if x can arrive z via y, x can also arrive z', $y < z' \le z$, via y. Namely, for each vertex x we must find the most distant vertex that it can arrive through a vertex y. Then all the vertices in between can all be reached by x through y. Assume the rightmost maximal clique containing x is Ci, (*rightmost-clique*(x) = i) u is the rightmost interval in Ci, (*rightmost-interval*(i) = u) v is the interval in Ci with the maximum *rightmost-clique* value j, (*clique-link*(i) = j, $i \le j$, *clique-link-interval*(i) = v) and the rightmost interval in Cj is u'. (*rightmost-interval*(j) = u', $u \le u'$, "=" holds if and only if $i = j = r$) Then, $D(x, u) = (1,0)$ and $D(x, y) = (1, 0)$ for all y, $x < y \le u$. Also, $D(x, v) = (1, 0)$ and x can arrive u' via v. This implies $D(x, u') = (2, v)$ and $D(x, y') = (2, v)$ for all y' with $u < y' \le u'$. The same procedure is repeated until $D(x, n)$ is computed. By the principle of optimality, the distance found is the shortest distance. This completes the proof. □

Now, we analyze the complexity of Algorithm 2. Finding *rightmost-clique* needs $O(rn)$ processors and the use of parallel prefix needs $O(\log r)$ time where r is the number of maximal cliques. Thus, Step 1 needs $O(n^2)$ processors and $O(\log n)$ time. Step 2 needs $O(r|C|)$ processors and the finding of *clique-link* using binary tree needs $O(\log |C|)$ time where $|C|$ is the number of intervals of maximum clique. Step 4 needs $O(n^2)$ processors and the computing of distance using parallel prefix needs $O(\log n)$ time, and the

operations of shift and reset need constant time. As for Step 3, it needs $O(r^2)$ processors. Since $D(1, n)$ is the diameter of interval graphs, the *rightmost interval string* of interval 1 has k elements where $k = D(1,n)$. So Step 3 needs $O(\log k)$ time to find the *rightmost interval string*. (Procedure of *rightmost interval string* is the application of recursive dobuling[11].) Hence, Algorithm 2 needs $O(n^2)$ processors and $O(\log n)$ time. If we apply a special technique[3, 17], then only $O(n^2 / \log n)$ processors are needed.

4. Conclusion

In this paper, we propose a new $O(n \log n)$ algorithm to find the maximal cliques of an interval graph, provided the interval model is given, and solve some problems such as cut vertices, bridges, and all-pair shortest path form maximal cliques. Parallel algorithm for the maximal cliques can be done in $O(\log n)$ time using $O(n^2)$ processors and all-pair shortest path prbolcm can also be solved in $O(\log n)$ time using $O(n^2)$ processors in parallel. Possible future researches include reducing the processor complexity of the parallel algorithm of maximal cliques and finding the edge connectivities of an interval graph by investigating the properties about the maximal cliques.

References

[1]. Alan A. Bertossi, On the domatic number of interval graphs, *Information Processing Letters* 28 (1988) 275-280.

[2]. R. Cole, Parallel merge sort, *SIAM Journal on Computing* 17(1988) 770-85.

[3]. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms* (The MIT Press, 1990).

[4]. M.C. Golumbic, *Algorithmic Graph Theory and Perfec: Graphs* (Academic Press, New York, 1980).

[5]. Tzong-Wann Kao and Shi-Jinn Horng, Computing k-vertex connectivity on an interval graph, *Technical report, Dept. of Elec. Eng. Nati. Taiwan Inst. of Tech, Taipei, Taiwan, ROC* (1994).

[6]. R. Mahesh, C. Pandu Rangan, and Aravind Srinivasan, On finding the minimum bandwidth of interval graphs, *Information and Computation* 95(1991)218-224.

[7]. Madhav V. Marathe, R. Ravi, and C. Pandu Rangan, An optimal algorithm to solve the all-pair shortest path problem on interval graphs, *Networks* 22(1992) 21-35.

[8]. Glenn K. Manacher. Terrance A. Mankus and Carol Joan Smith, An optimum $\Theta(n \log n)$ algorithm for finding a canonical hamiltonian path and a canonical hamitonian circuit in a set of intervals, *Information Processing Letters* 35(1990), 205-211.

[9]. J. Mark Keil, Finding hamiltonian circuits in interval graphs. *Information Processing Letters* 20(1985) 201-206.

[10]. C.U. Mertel and D. Gusfield. A fast parallel quicksort algorithm. *Information Processing Letters* 30(1989) 97-102.

[11]. Jagdish J. Modi, *Parallel Algorithms and Matrix Compution* (Clarendon Press, Oxford, 1988).

[12]. Stephan Olariu, A simple linear-time algorithm for computing the center of an interval graph, *Intern. J. Computer Math* 34(1990) 121-128.

[13]. Stephan Olariu, An optimal greedy heuristic to color interval graphs, *Information Processing Letters* 37(1991) 21-25.

[14]. Stephan Olariu, James L. Schwing, and Jingyuan Zhang, Optimal parallel algorithms for problems modeled by a family of intervals, *IEEE Trans. on Parallel Distrib. Systems* 3(1992) 364-74.

[15]. A. Srinivasa Rao and C. Pandu Rangan, Linear algorithm for domatic number problem on interval graphs, *Information Processing Letters* 33(1989/90), 29-33.

[16]. Alan P. Sprague and K.H. Kulkarni, Optimal parallel algorithms for finding cut vertices and bridges of interval graphs, *Information Processing Letters* 42(1992), 229-234.

[17]. Hong Shen, A fast parallel algorithm for integer sorting, *Parallel Computing 89. Proceedings of the International Conference* p. 331-6.