Distributed Algorithms for Chordal and Comparability Graphs and Their Subclasses

Daniel J. McCormick[1] **and Yevgeniy Vorobeychik**[2]

1 **Vanderbilt University, Nashville, TN, USA**
`daniel.mccormick@vanderbilt.edu`

2 **Vanderbilt University, Nashville, TN, USA**
`yevgeniy.vorobeychik@vanderbilt.edu`

Abstract

This paper contains distributed algorithms for recognition of chordal and comparability graphs, as well as algorithms for construction of a model of one intersection of these classes, interval graphs. It also includes a distributed Lexicographic Breadth First Search (LBFS) algorithm. LBFS has a wide range of sequential applications beyond these graph classes and could be a powerful tool for other distributed graph algorithms.

We show that distributed LBFS can be done with $O(n)$ messages per round, $O(n\log(n))$ bits per round, and $O(L)$ rounds, where L is the number of leaders it needs to split on. Next, we show that distributed chordal graph recognition can be done in $O(n)$ rounds with $O(n+m)$ messages per round and $O(n\log(n))$ bits per message. Third, we show that distributed comparability graph recognition can be done in $O(m)$ rounds with $O(n+m)$ messages per round and $O(n\log(n))$ bits per message. Finally, we show that interval graph construction can be done in $O(m)$ rounds with $O(n+m)$ messages per round and $O(n\log(n))$ bits per message.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases Chordal, Comparability, Lexicographic, Distributed, Graphs

Digital Object Identifier 10.4230/LIPIcs.DISC.2016.1

Eligible for best student submission

1 Introduction

Solving various problems on classes of graphs has been an area of significant study in Computer Science. There has been significant work, in particular, on 6 core problems: recognition, counting, representation, construction, characterization, and optimization [11]. Of these, recognition, construction, and optimization all have various serial algorithms that can be run on certain classes of graphs to produce output. In this paper, we will examine recognition and construction.

▶ **Definition 1.** *Recognition* is the problem of determining whether a graph belongs to a given class.

▶ **Definition 2.** *Construction* is the problem of taking a graph with the promise that it is in a certain class and building a model of the graph that makes solving problems on that graph easier.

Recognition can also often be done by constructing a model with a construction algorithm and then checking that the model is valid, though this is expensive or unsolved for certain models. We will look at both recognition and construction in the context of 2 classes of graphs and one intersection of these classes.

▶ **Definition 3.** The *chordal graph* class is the set of graphs without a chordless cycle of length greater than or equal to 4.

Fulkerson and Gross showed that a graph is chordal if and only if its vertices can be ordered in a perfect elimination scheme [3].

▶ **Definition 4.** A *perfect elimination scheme* is an ordering of vertices in a graph such that each vertex in the ordering forms a clique with all neighbors that follow it in the ordering.

▶ **Definition 5.** A *transitive orientation* of an undirected graph can be defined as a directing of all edges in the graph such that no transitivity violations exist in the graph, where vertices x, y, and z form a transitivity violation if $x- > y$ and $y- > z$ exist but $x- > z$ does not exist.

▶ **Definition 6.** The *comparability graph* class is the set of graphs with a valid transitive orientation.

Finally, we will also look at the intersection of the chordal graph class and the complement of the comparability graph class. Lekkerkerker and Boland showed that this class is equivalent to the class of interval graphs [6].

▶ **Definition 7.** The *interval graph* class is the set of graphs that can be modeled by the intersection of intervals on a line, where vertices are connected if and only if their corresponding intervals overlap on the line.

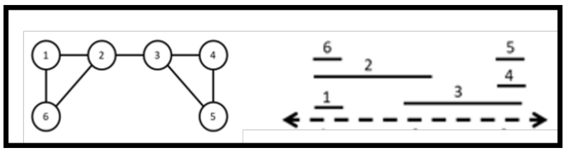

Figure 1 Example Interval Graph and Model

2 Computation Model

For the purposes of this paper, we will assume a connected graph with a synchronous model that closely aligns with the LOCAL model. In the LOCAL model, the graph is modeled with vertices represented as nodes and edges represented as bidirectional communication links, where each vertex can communicate along any edge via message passing. This message passing is assumed to be synchronized into rounds. During each round, every vertex can perform three steps: it can send a message to each of its neighbors, it can receive a message sent in the previous round, or it can perform computation [9].

In addition, under the LOCAL model there are several simplifying assumptions. First, nodes are considered to be infinitely powerful, so computation performed each round is assumed to be done instantly[9]. This assumption is reasonable in the context of this paper since the computation steps in each algorithm described are either constant time or dominated by the time spent sending, receiving, or processing messages. Second, the system is assumed to be fault-free [9]. In practice, this may or may not be true, but systems may have built in fault tolerance mechanisms to compensate for this. Finally, it is assumed that every vertex has a unique ID and that the ID is of size $O(n)$ [9]. We will assume that every vertex is aware of all of its neighbors' ids (which can easily be learned in a single round if necessary).

The only key difference from this model is that we will allow for the aggregation of data by a central synchronizer. This can be done at the end of any round, and allows all data to be centrally aggregrated by some function. The output of this function is then broadcast to all vertices. This can be done with a dedicated synchronizer node [8] or a spanning tree [5]. For this paper, we will assume that this is done with a dedicated synchronizer node.

To evaluate the performance of algorithms, we will use three metrics: the maximum number of rounds before the algorithm terminates, the maximum number of messages per round, and the maximum number of bits per message.

3 Lexicographic Breadth First Search

In order to solve the recognition and construction problems presented in this paper, we will rely heavily on a technique called Lexicographic Breadth First Search (LBFS). There are a number of variations of LBFS used for various applications, some of which we will discuss in this paper. We will now examine a generic sequential LBFS algorithm and propose a generic method for distributed LBFS.

3.1 Sequential LBFS

Sequential LBFS is a partitioning algorithm originally proposed by Habib, McConnell, Paul, and Viennot [4]. In general, the algorithm consists of 2 major parts: generating the initial partition and splitting of sets by active vertices. For the generic version, we will assume that the inital partition has already been performed and divided vertices into sets. The sets are ordered from smallest set number to largest. We will also assume we have been given some protocol P for dividing the sets correctly. The algorithm proceeds as described in Algorithm 1.

```
Data: A set of vertices $V$, an ordering of sets $O$, some protocol $P$
Mark some vertices as active and others inactive according to $P$.
while there exists an active vertex do
    Select an active vertex $v$ and potentially move it in $O$ according to $P$.
    for each set $S$ in $O$ do
        Divide $S$ into sets $S_n$ and $S_m$, with neighbors of $v$ being placed into $S_n$ and
          non-neighbors of $v$ being placed into $S_m$.
        if $v$ is not in the same set as $S$ then
            Replace $S$ in $O$ with $S_n$ and $S_m$, such that $S_m$ is placed closer to $v$ than
              $S_n$.
        end
    end
    Mark $v$ as inactive
    Possibly mark some vertices as active according to $P$.
end
if some set $S$ contains multiple vertices then
    Recursively run LBFS on $S$.
end
```

Algorithm 1: Generic Lexicographic Breadth First Search

Once this algorithm has been run, all vertices will be contained in their own sets. In addition, it results in useful properties when the correct protocol is used. As can be seen, the protocol controls when vertices are marked active or inactive and determines whether the leader should be moved in the ordering of sets.

3.2 Proposed Distributed LBFS

To distribute the generic LBFS algorithm, we will again begin with vertices somehow partitioned into sets, with each vertex aware of its own set number. In addition, each vertex will mark itself as either active or inactive according to some protocol P. From here, the algorithm is divided into recurring cycles of splitting steps and reduction steps. In between, the synchronizer plays 2 roles. Before splitting steps, it receives all active vertices and their set numbers, selects a leader according to P, and broadcasts that leader to all the vertices. Before the reduction steps, it receives all set numbers, removes duplicates, and broadcasts them to all vertices. The process of repeated splitting and reduction steps continues until the synchronizer receives no messages from active vertices. At this point, all vertices send their set numbers to the synchronizer. If any sets contain multiple vertices, the algorithm is run recursively on them. Both splitting and reduction steps are outlined below.

```
Data: A vertex $v$ with a set number $s$ and some protocol $P$
$v$ receives message from synchronizer with the id of the leader for the round $L$ and its
 set number $s'$.
if ($s > s'$ and $L$ is a neighbor of $v$) or ($s < s'$ and $L$ is not a neighbor of $v$) then
    $s := 2s + 1$
else
    $s := 2s$
end
if $v$ is the leader then
    $v$ marks itself inactive and assigns itself to set -1
end
$v$ possibly marks itself as active according to $P$
$v$ sends its set number to the synchronizer
```

Algorithm 2: Splitting Step of Generic Distributed LBFS

▶ **Lemma 8.** *If the protocol's used are the same, running Algorithm 1 and running Algorithm 2 until all vertices are inactive will produce the same ordering of sets, where sets are ordered from left to right at the end of Algorithm 1 and from smallest to largest at the end of Algorithm 2.*

Proof. Since the initial ordering and active status is determined by the protocol, both orderings will initially be the same, and the leaders selected by the protocol will be the same. Both algorithms then divide sets into 2 and place the neighbors further from the leader. Finally, the leader will be marked inactive in both algorithms and the protocol may mark some other vertices active. Since the running of these algorithms is virtually identical, they must produce the same ordering of sets. ◀

▶ **Lemma 9.** *The reduction step described in Algorithm 3 will not change the ordering of sets and will result in all set numbers being less than or equal to n.*

```
Data: A vertex $v$ with a set number $s$
$v$ receives message from synchronizer containing a set of unique numbers $S$
$i := 0$
for each number $x$ in $S$ do
    if $x \leq s$ then
        $i := i + 1$
    end
end
$s := i$
if $v$ is active then
    $v$ sends its identifier and set number to the synchronizer
end
```

Algorithm 3: Reduction step of distributed LBFS

Proof. In the reduction step, each vertex counts the number of sets that are less than its own and sets its own set number to that number plus 1. Given 2 vertices x and y with set numbers $s_x < s_y$, the number of set numbers less than s_y must be greater than the number of set numbers less than s_x, so s_x must still be less than s_y after the reduction. Since only $n - 1$ vertices can have set numbers less than s_y, s_y will have a set number of at most n. ◀

▶ **Theorem 10.** *Using distributed LBFS with protocol P will produce the same ordering of sets as using Algorithm 1 with P and will use $O(n)$ messages per round, $O(n \log(n))$ bits per message, and $O(L)$ rounds, where L is the number of leaders that are chosen by P.*

Proof. By Lemma 8, running Algorithm 2 repeatedly will produce the same results as running Algorithm 1. Since Algorithm 3 doesn't change the ordering of sets by Lemma 9, running the entire distributed LBFS algorithm must produce the same results as running Algorithm 1.

Next, since each vertex only needs to communicate with the synchronizer in both steps, each vertex will only need to send and receive 1 message per round for a total of $O(n)$ messages. In the splitting step, each message contains only an id and set number. Since each of these have size $O(n)$, this will use $O(\log(n))$ bits per message. In the reduction steps, however, the synchronizer sends messages containing the set numbers of all vertices. Since these are all of size $O(n)$, this means that there will be $O(n \log(n))$ bits per message. Finally, since a leader is chosen at the start of each splitting step and this corresponds to 1 splitting round and 1 reduction round, there will be at most $O(L)$ rounds. ◀

4 Chordal Graph Recognition

Recognition is an important problem since it lets us classify graphs into restrictive classes that can make solving problems easier. We will now look at recognition of chordal graphs.

4.1 Existing Sequential Algorithm

We will first examine a sequential algorithm for chordal graph recognition published by Rose, Tarjan, and Lueker [10]. The algorithm has 2 basic steps: LBFS and verification of the perfect elimination scheme. For LBFS, the following protocol is used: all vertices are initially partitioned into a single set and marked active. After this, no further vertices are marked active, and at the start of each pass the leader is selected randomly from the farthest right set containing active vertices. It has been proven that if a perfect elimination scheme exists, the ordering of vertices produced by LBFS will be one [10].

Next, we will use Algorithm 4 to verify that the perfect elimination scheme is valid. Algorithm 4 has been shown to return true if and only if the ordering is a perfect elimination scheme [10]. The whole process can be performed in $O(n+m)$ time.

```
Data: An ordering of vertices
for each vertex v in the ordering do
    S :=neighbors of v to the right of v in the ordering.
    for each combination of vertices x,y in S do
        if x does not have an edge to y then
            return false
        end
    end
end
return true
```

Algorithm 4: Verifying a perfect elimination scheme

4.2 Proposed Distributed Algorithm

The proposed distributed algorithm for chordal graph recognition builds off of the 2 step algorithm described above. Like that algorithm, it is divided into LBFS and verification of the perfect elimination scheme, but it is designed to be run on a distributed system.

To perform LBFS, we will use an identical protocol to the sequential version: all vertices are initially partitioned into a single set with set number 1 and marked active. After this, no further vertices are marked active, and at the start of each pass the leader is selected randomly from the largest set containing active vertices. By Theorem 10, this will produce the same ordering of sets as the sequential LBFS, so it will produce a perfect elimination scheme if one exists. Since the graph is chordal if and only if it has a perfect elimination scheme, it is chordal if and only if the output of LBFS is a perfect elimination scheme, which can be verified by Algorithm 5 below.

```
Data: An active vertex $v$ with a set number $s$
Send $s$ to all neighbors
Receive all incoming set numbers, throw out any with set numbers less than its own.
Aggregate identifiers of remaining messages into list $L$.
Send ARE_YOU_CLIQUE message containing $L$ to all vertices in $L$.
for each incoming message containing a list of identifiers $L'$ do
    for each identifier $i$ in $L'$ do
        if $v$ does not have an edge to vertex $i$ then
            Send NO message to synchronizer.
            return
        end
    end
end
Send YES message to synchronizer
```

Algorithm 5: Distributed verification of Perfect Elimination Scheme

▶ **Lemma 11.** *Algorithm 5 will result in all vertices sending a YES message to the synchronizer if and only if the vertices ordered from least set number to greatest form a perfect elimination scheme.*

Proof. Order all vertices from smallest set number to largest. Suppose all vertices send a YES message. Then each neighbor of a given vertex v that has a larger set number than it clearly must have edges to all other neighbors of v that have larger set numbers than v. Therefore, v forms a clique with all neighbors that come after in the ordering. Therefore, the ordering is a perfect elimination scheme.

Next, suppose that some vertex v sends a NO message. Let vertex x be the vertex that sent the ARE_YOU_CLIQUE message that triggered the NO. There must be some vertex that has a larger set number than x, is a neighbor of x, but isn't a neighbor of v. Therefore, x does not form a clique with all neighbors that come after it in the ordering, so the ordering is not a perfect elimination scheme. ◀

4.2.1 Complexity Analysis

▶ **Theorem 12.** *Using distributed LBFS and Algorithm 5, distributed chordal graph recognition can be done in $O(n)$ rounds, with $O(n + m)$ messages per round, and $O(n \log(n))$ bits per message.*

Proof. Since the distributed LBFS protocol never marks vertices active after the initial partition and leaders are marked inactive at the end of their leadership round, there can only be $O(n)$ leaders. Therefore, by Theorem 10, distributed LBFS will take $O(n)$ rounds, $O(n)$ messages, and $O(n \log(n))$ bits per message.

Next, clearly the verification step takes only 3 rounds since it only has to send and receive messages 3 times, so the verification takes $O(1)$ rounds. Since each vertex sends its set number to all neighbors, this requires that a message be sent along every edge. Likewise, each vertex must send a message to the synchronizer. Therefore, in total, there must be $O(n + m)$ messages per round. Finally, each message contains at most one set number and id, so there will be $O(n \log(n))$ bits per message. ◀

5 Comparability Graph Recognition

Now we will look at recognition of another natural graph class: comparability graphs.

5.1 Existing Sequential Algorithm

We will first look at the sequential algorithm published by Habib, McConnell, Paul, and Viennot [4]. Like the above chordal recognition algorithm, the comparability graph recognition algorithm consists of 2 steps which have been outlined below.

The first step uses LBFS to attempt to generate a transitive orientation of the graph. To generate an initial partition, a source vertex is chosen using Algorithm 6.

Once the source is selected, LBFS is performed with the following protocol: all vertices are partitioned into set 2 and marked inactive except for the source which is placed into set 1 and marked active. During each pass, vertices are marked active anytime one of their neighbors goes from being in the same set as them to a different set. The leader is selected randomly from all active vertices.

Once LBFS has been performed, edges are directed from left to right based on the ordering. It has been shown that if a transitive orientation exists, Algorithms 6 and LBFS with this protocol will produce one [4].

```
Data: A set of vertices $V$ and edges $E$
Choose a random vertex $v$ and place $v$ in non-candidate set $N$
Place all vertices except for $v$ into candidate set $C$ and mark them active
while $|C| > 1$ do
    if there exists an active vertex $w$ in $N$ then
        for each vertex $x$ in $C$ that is not a neighbor of $w$ do
            Mark $x$ as $READY\_TO\_MOVE$
        end
        if there exists a vertex in $C$ not marked $READY\_TO\_MOVE$ then
            for each vertex $x$ in $C$ marked $READY\_TO\_MOVE$ do
                Remove $x$ from $C$, place it in $N$, and mark it active
            end
        else
            Mark all vertices as $NOT\_READY\_TO\_MOVE$
        end
    else
        Move a random vertex $w$ from set $C$ to set $N$ and mark it active
    end
end
```

Algorithm 6: Source Selection for Comparability Graph Recognition

Verifying the transitive orientation has been shown to be equivalent in complexity to adjacency matrix multiplication, which can be used to generate all paths of length 2 [11]. These paths are then compared against the original adjacency matrix to ensure there are corresponding paths of length 1. This dominates the time complexity of the whole recognition algorithm, so recognition of chordal graphs can be done in time proportional to matrix multiplication, or $O(n^{2.81})$ time [11].

5.2 Proposed Distributed Algorithm

The proposed distributed algorithm is also divided into LBFS and verification steps. This version of LBFS is similar to the sequential version. To select a source, all vertices are initially assigned to a candidate set except for the vertex with the smallest id, which is assigned to the non-candidate set. All vertices mark themselves as active. Next, the Algorithm 7 is performed repeatedly.

While each vertex is performing this algorithm, the synchronizer receives the messages and chooses a leader from the non-candidate set and broadcasts this to all vertices. If at any point there are no candidate set vertices that send messages, the synchronizer adds the $REVERSE$ attachment. If only one vertex sends a message from the candidate set, the synchronizer terminates the algorithm. If there are no vertices that send messages from the non-candidate set, the synchronizer chooses a vertex from the candidate set.

▶ **Theorem 13.** *Algorithm 7 will produce the same source vertex as Algorithm 6 if the randomly chosen leaders are chosen in the same way.*

Proof. Assume the leaders chosen are the same. From here, the algorithms work virtually the same way: neighbors of the leader are assigned to the non-candidate set until exactly 1 vertex remains in the candidate set, with the same constraints for edge cases. Therefore, it will produce the same source. ◀

```
Data: An active vertex $v$ in a candidate or non-candidate set
$v$ receives message from synchronizer containing identifier of leader $L$.
if message contains $REVERSE$ attachment and $v$ has marked itself $TENTATIVE$
 then
    $v$ moves itself to the candidate set
end
$v$ marks itself $NOT\_TENTATIVE$
if $v = L$ then
    $v$ assigns itself to the non-candidate set
else
    if $L$ is a neighbor of $v$ and $v$ is in the candidate set then
        $v$ assigns itself to the non-candidate set $v$ marks itself TENTATIVE
    end
    $v$ sends its id and set number to the synchronizer
end
```

Algorithm 7: Source selection of modified distributed LBFS

Next, LBFS is performed using the same protocol as sequential comparability recognition: all vertices are partitioned into set 2 and marked inactive except for the source which is placed into set 1 and marked active. During each pass, vertices are marked active anytime one of their neighbors goes from being in the same set as them to a different set. The leader is selected randomly from all active vertices. By Theorem 10, this is guaranteed to produce the same ordering as the sequential version. Therefore, if a transitive orientation exists, ordering the edges from least set number to greatest will produce it.

Verifying that the transitive orientation is valid can be done using Algorithm 8.

```
Data: A vertex $v$ with a list of outgoing edges $O$ and list of incoming edges $I$
for each vertex $x$ in $I$ do
    Send all vertices in $O$ to $x$
end
for each incoming list of vertices $O'$ do
    for each vertex $x$ in $O'$ do
        if $x$ is not in $O$ and $x! = v$ then
            Send $NO$ message to synchronizer return
        end
    end
end
Send $YES$ message to synchronizer
```

Algorithm 8: Distributed Transitive Orientation Verification

▶ **Lemma 14.** *A transitive orientation is valid if and only if all nodes send a YES message to the synchronizer when Algorithm 8 is run.*

Proof. Assume all vertices send a YES message to the synchronizer. Then for each set of vertices x, y, z with edges $x- > y- > z$, x must have received y's list of outgoing edges and verified that it had an edge to all of those vertices, including z, so $x- > z$ must exist. Therefore, the transitive orientation is valid.

Next, assume some vertex v did not send a YES message (and by extension sent a NO message). Then there must be some vertex y with an outgoing edge $x- > y$ such that x does not have an edge to some vertex z with $y->z$ and $x!=z$. Therefore, this is a transitivity violation and the transitive orientation is not valid. ◀

5.2.1 Complexity Analysis

▶ **Theorem 15.** *Using LBFS and Algorithm 8, comparability graph recognition can be done in $O(m)$ rounds with $O(n+m)$ messages per round and $O(n \log(n))$ bits per message.*

Proof. First, we will analyze the LBFS step. Finding the source clearly takes $O(n)$ rounds since one vertex is marked inactive per round, has a message complexity of $O(n)$ since only synchronization messages are used, and has $O(\log(n))$ bits per message since each message only contains the id and set number of one vertex. Each subsequent set of rounds moves a vertex from active to inactive. Since a vertex can only become active when its neighbor moves to a different set, all vertices combined can become active at most $O(m)$ times, so this step has $O(m)$ rounds. By Theorem 10, LBFS will use $O(m)$ rounds, $O(n)$ messages per round, and $O(n \log(n))$ bits per message.

Next, we will analyze the verification step. This requires $O(1)$ rounds since messages must only be received twice. In addition, a message is sent along each edge exactly once excluding synchronization messages, so this step has a message complexity of $O(m)$. Each message contains at most n ids of size at most n so there are $O(n \log(n))$ bits per message. ◀

6 Interval Graph Construction

Next, we will look at the problem of constructing a model for interval graphs.

6.1 Existing Sequential Algorithm

For this paper, we will examine the sequential algorithm developed by Habib, McConnell, Paul, and Viennot [4]. First, find a transitive orientation of the complement of the graph G. Next, get a perfect elimination ordering of G. Then, use Algorithm 9 to create a model [4].

```
Data: set of vertices ordered in perfect elimination scheme $V$ and set of directed
      edges from the complement $E$
Define a set of maximal cliques $S := \{\}$
for each vertex $v$ in $V$ going left to right do
    $C := \{v\} + \{$all neighbors of $v$ that appear after $v$ in $V\}$
    if $C$ is not contained in any other maximal cliques in $S$ then
        $S := S + \{C\}$
    end
end
Define an ordering of cliques $O := []$
while $S$ is not empty do
    Select a clique $C$ from $S$ that has only outgoing edges in $E$
    Append $C$ to $O$ and delete $C$ from $S$
end
for each vertex $v$ in $V$ do
    $INDEX\_SET$ :=set of indices in $O$ of all cliques containing $v$
    Interval of $v := [min(INDEX\_SET) - 0.1, max(INDEX\_SET) + 0.1]$
end
```
Algorithm 9: Sequential Interval Construction

This algorithm can be performed in $O(n)$ time [4].

6.2 Proposed Distributed Algorithm

Like the sequential algorithm, the proposed distributed algorithm builds off of previously described techniques. First, find a perfect elimination scheme of the graph and at each vertex store whether each of its neighbors come before or after it in the ordering. Next, to find the maximal cliques, perform Algorithm 10 at each vertex:

```
Data: A vertex $v$, a set of its neighbors $N$, and an ordering of $\{v + N\}$, $O$
Define a set of vertices $C := \{v\}$
for each vertex $x$ that comes after $v$ in $O$ do
    Add $x$ to $C$
end
Send $C$ to all neighbors of $v$
for each incoming message containing clique $C'$ do
    if $C \subset C'$ then
        $C := C'$
    end
end
Store $C$ as $v$'s clique
Send $C$ to all neighbors of $v$
for each incoming message from vertex $x$ containing clique $C'$ do
    Store $C'$ as $x$'s clique
end
```

Algorithm 10: Distributed Maximal Clique Construction

▶ **Theorem 16.** *At the end of Algorithm 10, all vertices will have stored a maximal clique.*

Proof. Let C be the clique initially generated at vertex v. If C is maximal, it will not ever be replaced. Suppose C is not maximal, so there exists some vertex x such that $C + x$ is a clique and $x \not\subset C$. Since C was generated by looking at all vertices that come after it in the perfect elimination ordering, x must come before v in the ordering.

Therefore, since all other vertices in C must come after v, x must have a maximal clique C' such that $C \subset C'$, so C' will replace C at vertex v and v will contain a maximal clique. ◀

After this, we will perform another partitioning algorithm. Like previous versions, there will be splitting and reduction rounds. Unlike previous rounds, however, we will be focusing on dividing maximal cliques into sets instead of individual vertices.

First, to establish the initial partition, we will first take a transitive orientation of the complement. To avoid reloading the complement of the graph, we will perform the LBFS algorithm described in the comparability graph recognition section on the original graph G with a slight modification: each node will treat its neighbors as non-neighbors and vice versa. Since the algorithm doesn't require messages to be passed to neighbors, this will not disrupt it. Next, each vertex will send its set number to all neighbors and perform Algorithm 11.

▶ **Theorem 17.** *If G is an interval graph, running Algorithm 11 at all vertices of G will result in exactly one maximal clique that has only outgoing edges in a transitive orientation of the complement being marked the source*

```
Data: A vertex $v$ with set number $s$, edges $E$, and associated maximal clique $C$
Receive set numbers from neighbors, store in list $S'$
$CANDIDATE$ :=true
for $i = 1; i < s; i++$ do
    if $i \not\subset S'$ then
        $CANDIDATE$ :=false
    end
end
if $CANDIDATE$ then
    Send $CANDIDATE$ message to all neighbors
end
Receive incoming messages from neighbors, store neighbors that sent message as $M$.
$SOURCE$ :=true
if $C \not\subset M$ then
    $SOURCE$ :=false
end
```

Algorithm 11: Clique Source Selection for Interval Graph Construction

Proof. Recall, all edges from any maximal clique C to another maximal clique C' must be oriented in the same direction in the transitive orientation of the complement of an interval graph and they can be ordered based on this orientation [4]. Since there must be one maximal clique that comes first in the ordering, this clique must have all edges oriented outwards.

Assume some maximal clique C is this source. Then, if there is an edge in the complement going from vertex v with set number s to another vertex x, x must have a higher set number than s. Therefore, all set numbers less than s must be associated with vertices that x doesn't have edges to in the complement (and thus does have edges to in G). Since the reduction guarantees that there will be set numbers from 1-n and s must be less than n by Lemma 9, any positive set number $t < s$ must be associated with a neighbor of v, so v will mark itself as a candidate. Since all vertices in C must be candidates, they'll mark C as a source.

On the other hand, suppose C' is not a source clique. Then there must be some edge in the complement from a vertex x with set number s' to a vertex v in C' with set number s with $s' < s$. Since this edge doesn't exist in G, and no other vertex can have set number s', v must mark itself as not a candidate, so C' will not mark itself as a source. ◀

Once a source clique C has been chosen, each vertex that has assigned itself clique C assigns itself to set 1. All other vertices assign themselves to set 2.

Once this initial partition has been formed, we will continue to partition the cliques with a technique similar to modified LBFS. Initially, only vertices in set 1 will mark themselves as active. Before each splitting step, the synchronizer chooses an active vertex and sends its associated clique and set number to all vertices in the graph. Then the splitting step described in Algorithm 12 is repeatedly interleaved with the reduction step of Algorithm 3.

If at any point no vertices are active, each vertex sends its clique and set number to the synchronizer. For each set number s that contains multiple distinct cliques, all vertices with set number equal to s are marked for recursion and Algorithm 13 is run on that set, followed by the same cycle of splitting and reduction steps using Algorithms 14 and 4. If all vertices are marked inactive and the synchronizer doesn't mark any vertices for recursion, we move on to the assignment of intervals by Algorithm 15.

▶ **Theorem 18.** *If G is an interval graph and all maximal cliques are represented at a vertex*

```
Data: An active vertex $v$ with a clique $C$ and set number $s$
Receive synchronizer message with vertex $x$, clique $C'$, and set number $s'$
if ($s > s'$ and ($C$ and $C'$ don't contain common vertices)) or ($s < s'$ and ($C$ and $C'$
  contain common vertices)) then
    $s := 2s + 1$
else
    $s := 2s$
end
if $v = x$ then
    $v$ marks itself inactive
end
Send $s$ to neighbors
Receive incoming set numbers $S$
for each set number $s'$ in $S$ do
    if $s! = s'$ and $s$ was equal to $s'$ before the previous round then
        $v$ marks itself as active
    end
end
```

Algorithm 12: Distributed Ordering of Cliques for Interval Graph Construction

in G with a correctly identified source clique, running Algorithms 12 and 3 repeatedly will result in ordering of cliques in which vertices only appear in consecutive cliques.

Proof. It has previously been shown that if G is an interval graph, such an ordering exists [4]. Since the source clique is placed at the start of the ordering and never moved, the initial partition is consistent with one such ordering [4]. Next, each splitting places cliques with neighbors of the splitting clique C closer to C. If this was not done, there would be some vertex v that is contained in C and is contained in some clique C_1 but not in C_2, but C_2 would be between C and C_1. This violates the target ordering, so all neighbors must be placed in the set adjacent to C and each subsequent splitting step doesn't create an inconsistency. By induction, no splitting step causes an inconsistency.

Next, when recursion is performed, all cliques with the recursion being performed on them are in the same set s. Therefore, for each clique C not in s, either all cliques in s have a shared vertex with C or all cliques in s don't have a shared vertex with C. Therefore, splitting this set will not cause any inconsistencies in the ordering between vertices in s and outside of s. Since the recursion doesn't produce any inconsistencies, and the splitting and reduction don't produce inconsistencies, the entire algorithm won't produce inconsistencies. ◀

```
Data: An active vertex $v$ with a clique $C$ and set number $s$
$v$ sends out $s$ to all vertices in $C$
$v$ receives all incoming set numbers, stores in set $S$
$v$ assigns itself the interval $(min(S) - 0.1, max(S) + 0.1)$
```

Algorithm 13: Distributed assignment of intervals

▶ **Theorem 19.** *Let G be a graph with an ordering of all maximal cliques O, where for each vertex v in G, v only appears in consecutive cliques in O. Running Algorithm 13 on G will produce correct intervals corresponding to an interval graph representation.*

Proof. Since each vertex is only part of consecutive maximal cliques, it will have edges to all vertices that appear in those cliques and no other vertices. Therefore, by assigning each vertex an interval that spans that distance, we are guaranteeing that vertices' intervals overlap only if they share a maximal clique. Thus, by definition, this is an interval graph. ◀

6.3 Complexity Analysis

▶ **Theorem 20.** *Using the above algorithms, interval graph construction can be done in $O(m)$ rounds, with $O(n+m)$ messages per round, and $O(n \log(n))$ bits per message.*

Proof. Finding the perfect elimination scheme takes $O(n)$ rounds, $O(n+m)$ messages per round, and $O(n \log(n))$ bits per message. Next, finding the maximal cliques takes $O(1)$ rounds with a maximal clique sent along every edge exactly once, so it takes $O(m)$ messages per round, and $O(n \log(n))$ bits per message. Finding the initial partition is dominated by modified LBFS, so there are $O(n+m)$ rounds, $O(m)$ messages per round, and $O(n \log(n))$ bits per message.

Similarly, the partitioning algorithm works essentially the same way as the original LBFS algorithm, with vertices each being marked inactive at most once per outgoing edge, at most one message being sent per edge per round with at most n set numbers contained in it, so there are $O(m)$ rounds, $O(m)$ messages per round, and $O(n \log(n))$ bits per message.

Finally, assigning intervals has 1 round where a set number is sent along every edge once. Therefore, it has $O(1)$ rounds, $O(m)$ messages per round, and $O(\log(n))$ bits per message.

Synchronization messages for the whole algorithm clearly require at most $O(n)$ messages per round and $O(n \log(n))$ bits per message. Therefore, aggregating across all steps there are $O(m)$ rounds, $O(n+m)$ messages per round, and $O(n \log(n))$ bits per message. ◀

7 Related Work

This paper presents the first distributed algorithms for solving problems on the chordal and comparability graph classes or subclasses. There has been significant related work on sequential algorithms for chordal and comparability graph algorithms, including sequential algorithms for chordal graph recognition by Rose, Tarjan, and Lueker [10], as well as comparability graph recognition and interval graph construction by Habib, McConnell, Paul, and Viennot [4].

In addition, there has been significant work done on distributed graph problems on other classes. Notable examples include various optimization algorithms on planar graphs by Czygrinow, Hańćkowiak, and Wawrzyniak [2] and distributed domination on classes of bounded expansion by Amiri, Mendez, Rabinovich, and Siebertz [1]. There has also been a significant amount of work on approaching similar problems using MapReduce, including work by Lin and Schatz [7], an approach that could yield effective algorithms for these classes, though likely with signficantly more messaging overhead [8].

8 Conclusion

In this paper, we proposed algorithms for solving the distributed recognition problem on chordal graphs with $O(n)$ rounds, $O(n+m)$ messages per round, and $O(n \log(n))$ bits per message and recognition on comparability graphs with $O(m)$ rounds, $O(n+m)$ messages per round, and $O(n \log(n))$ bits per message, as well as the distributed construction problem on interval graphs with $O(n+m)$ rounds, $O(m)$ messages per round, and $O(n \log(n))$ bits per message. In the process, we came up with an algorithm for parallelizing Lexicographic Breadth First Search, a technique which is used in several graph algorithms beyond just these classes [10], with $O(n)$ messages per round, $O(n \log(n))$ bits per round, and $O(L)$ rounds, where L is the number of leaders it needs to split on..

References

1 Saeed Amiri, Patrice Mendez, Roman Rabinovich, and Sebastian Siebertz. Distributed domination on graph classes of bounded expansion. *CoRR*, abs/1702.02848, 2017. URL: `http://arxiv.org/abs/1702.02848`, `arXiv:1702.02848`.

2 Andrzej Czygrinow, Michal Hańćkowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In *Proceedings of the 22Nd International Symposium on Distributed Computing*, DISC '08, pages 78–92, Berlin, Heidelberg, 2008. Springer-Verlag. URL: `http://dx.doi.org/10.1007/978-3-540-87779-0_6`, `doi:10.1007/978-3-540-87779-0_6`.

3 Delbert Ray Fulkerson and Oliver Gross. Incidence Matrices and Interval Graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965. `doi:10.2140/pjm.1965.15.835`.

4 Michel Habib, Ross McConnell, Christophe Paul, and Laurent Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2):59–84, 2000. URL: `http://perso.prism.uvsq.fr/~qst/Tarjan/Lex-BFS.pdf`.

5 K.E. Johansen, U.L. Jørgensen, S.H. Nielsen, S.E. Nielsen, and S. Skyum. *A distributed spanning tree algorithm*. Springer, Berlin, Heidelberg, 2005.

6 C. Lekkeikerker and J. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51(1):45–64, 1962. URL: `http://eudml.org/doc/213681`.

7 Jimmy Lin and Michael Schatz. Design patterns for efficient graph algorithms in mapreduce. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, MLG '10, pages 78–85, New York, NY, USA, 2010. ACM. URL: `http://doi.acm.org/10.1145/1830252.1830263`, `doi:10.1145/1830252.1830263`.

8 Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM. URL: `http://doi.acm.org/10.1145/1807167.1807184`, `doi:10.1145/1807167.1807184`.

9 David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, 2000.

10 Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput*, 5(2):266–283, 1976. URL: `http://epubs.siam.org/doi/abs/10.1137/0205021`.

11 Jeremy P. Spinrad. *Efficient Graph Representations*. Fields Institute Monograph, 2003.