Distributed algorithm for finding maximal clique cover and Leader Election in interval graphs

Sai Ganesh
email : ssai@wipsys.soft.net *

September 20 1998

Abstract

Centralized approach of maintaining and controlling a collection of nodes in a distributed system has attracted wide attention because of thier ease of implementation, maitenance and effectiveness. The leader election is a problem in which all the nodes of the network agree on consensus to elect a node which may be responsible for functionalities common in the distributed network. Functionalities such as coordinating various events in the system, resource allocation and timed execution in solving a critical problem are some of common task handled by the elected centralized leader.

This paper focusses on the set of graphs called the interval graphs (the definition of which follows) in which efficient techniques to achieve distributed maximal clique cover and leader election problem are provided. The best case upper bound on the messaging complexity for leader election algorithm are less than the proven lower bound for the problem for general anonymous distributed networks.

Given a finite number of intervals on a straight line, an interval graph G is such that all the intervals form the vertices of the graph and there is an edge between node i and j if and only if the interval i has atleast partial overlap with the interval j on the straight line. Application of interval graphs vary from graph-coloring problem to seriation of set of items in chronological order to the process scheduling mechanism in operating systems.

*I like to express my thanks to Meena and KamaKoti both of MatScience, Chennai for giving me the excellent oppurtunity to work at MatScience and for greatly assisting this work

Using some of the interesting properties of the interval graphs given in the sections following, we achieve an overall best case messaging complexity of $O(n)$ for finding the maximal clique cover (a set of vertices) in the network and electing the leader.

1 Introduction

Leader election is a fundamental problem in distributed systems. The elected leader of a network may be responsible for coordinating various events in the system or solving a critical problem. The problem involves in that given a set of prospective candidates, the technique is to be call the node n_i with the extreme identification id_i to be the leader of the network.

Consider an undirected, connected network $G = (V, E)$ which has a total of number of processes $n =\mid E \mid$ in the system. Consider an interval graph defined from a set of given intervals on a straight line. Interval graphs can be defined more formally as below[Golumbic]:

An undirected graph $G = (V, E)$ is an interval graph if its vertices can be placed into one-to-one correspondence with a set of intervals $\Im$ of a linealy ordered set (like the real line) such that two vertices are connected by an edge in G if and only if their corresponding intervals have nonempty intersection. We call $\Im$ an interval representation of G.

Making use of some of the assumptions and properties of permutation graphs as defined in the next section, this wave algorithm provides a best case messaging complexity of $O(n)$ and a worst case message complexity of $O(n^2)$.

An algorithm that elects a leader in general network with $O(\mid E \mid +n)$ messaging complexity is proposed by Gallager[Gallager]. It is shown that for a general graph, atleast $\Omega(\mid E \mid +n \log n)$ messages must be exchanged which is the lower bound to the problem of election in the worst case[KMZ]. Other algorithms with lesser messaging complexity for a specialized and restricted subsets of network topology are proposed[KMZ, DKR, Peterson].

Below are the listings of some of the properties of interval graphs that will be directly or indirecly assist in idea conception of the algorithm implementation :

1.1 Interval graph characteritics

1. [Gilmore and Hoffman] Let G be an interval graph. The following statements are equivalent :

1. hspace*9 G is an interval graph

2. hspace*9G satisfies triangulation properties in that it contains hspace*9no chordless 4-cycle and its complement $\overline{G}$ is hspace*9a comparability graph

3. hspace*9The maximal cliques of G can be linearly ordered such hspace*9that, for every vertex $x \in V$ in G, the maximal hspace*9cliques containing x occur consecutively

Alternatively, an undirected graph G is an interval graph if and only if G is a triangulated graph and its complement $\overline{G}$ is a comparability graph.

2. Efficient, linear-time algorithms are possible for a realization of an interval graph.

3. An induced subgraph of an interval graph is an interval graph

1.2 Triangulation graph characteritics

Since every interval graph is a triangulated graph (vice versa need not always hold), following are the properties/characteristics which assisted in conception of the idea for the algorithm implementation :

1. Let G be an undirected graph. The following statements are equivalent:

1. hspace*9G is a triangulated graph

2. hspace*9G has a perfect vertex elimination scheme σ. hspace*9Moreover, any simplical vertex v can start a perfect hspace*9scheme

2. For any triangulated graph G, it is possible to create a perfect elimination scheme σ with a time and space complexity proportional to $\mid V \mid + \mid E \mid$ i.e. of order $O(n)$ in a sequential machine

3. For any triangulated graphs G with a perfect elimination scheme σ, it is possible to calculate all the maximal cliques and the chromatic number in $O(\mid V \mid + \mid E \mid)$ in a sequential machine

2 Model and Assumptions

This section defines the model for the proposed computing along with the set of assumptions made of the system and algorithmic assumptions.

Consder an interval graph $G = (V, E)$ which is a finite, undirected, connected network. Following assumptions are made for working in this model.

1. Communication among the processes is asynchronous with no assumed timeout for messages to arrive at the destination node. However, total reliability with no loss of messages is assumed

2. Each node (we will be using the term node and process interchangeably) supports a FIFO order of queueing mechanism for the reception of messages

3. Each node n_i is idenfied by an identifier id_i where id_i denotes the unique identifier for that node.

4. Each node n_i identifies its neighbouring nodes/ports using their unique identifier (id_k k $\in$Neig$_i$)

5. Each node n_i communicates only with the nodes of depth 1 from itself i.e. only with its neighbouring nodes. However, decision at node n_i can take place based on this sequence

6. Exaclty one node spontaneously wake-up and initiate the whole process. This node starts sending sending events to the other nodes in the network

7. There is no limitation on the number of messages that can be sent and on the size of each of these messages

3 Algorithm

This sections proposes the algorithm for the leader election for the above defined permutation graphs.

3.1 General Outline

Each node holds the unique identifiers of all its neighbouring nodes and understands the port-to-id mapping. Exaclty one node starts the whole process spontaneously and starts sending messages to the other nodes. All other nodes wakeup upon reception of a message first runs its local initialization procedure before servicing the message received.

The algorithm proceeds in a sequence of five steps as described below.

1. Find the perfect elimination scheme σ for the given interval graph (a triangulated graph).

2. Construct a maximal clique set for the given graph (this is one the properties of triangulated graphs before)

3. Clique sets elect their own leader concurrently in $O(1)$ and understand the leader among themselves.

4. Each of the nodes n_{li} in the clique set communicate with their neighbour n_{lk} to check if it encounters a node of another clique set. If it does, it compares the identifier of its leader with that of the later. If n_{li} is less than that of n_{lk}, then this is taken to be the new leader. Once its finished searching for leader, it announces this new leader (if found) to its clique set nodes.

5. In this way, the overall leader is understood by all the nodes in all the clique sets of the network and need not be announced explicitly

3.2 Formal Description

Algorithm defined below contains an overall of 10 different steps before a leader is elected for the network.

Algorithm Elect$_{L}eader$
Output : $Attermination, eachnode\text{n}_i$ has identifier id_l of the leader

1. // Calculate perfect elimination scheme σ
   // **Initiator**
   $numbered_i^i$:= **true**;
   $label_i^i := \emptyset$;
   forall $k \in$Neig$_i$ **do**
   begin
   $label_i^k := \emptyset$;
   numbered$_i^i$:= **false**;
   end

   $received\_add_i := \emptyset$;
   $\sigma_i :=\ id_i$;
   $parent_i := id_i$;

   forall $k \in$Neig$_i$ **do**
   begin
   send add(σ_i) to node n_k;
   $label_i^k := label_i^k \cup$id$_i$;
   end

   send search(σ_i) for k, max$\{label_i^k \mid \forall k \in$Neig$_i \wedge$numbered$_i^k$ = **false**$\}$;
   $numbered_i^k$:= **true**;

2. // Non-initiators initialization
   $\sigma_i := \emptyset$;
   forall k$\in$Neig$_i$ **do**
   begin

$label_i^k := \emptyset$;
numbered$_i^k$:= **false**;
end

3. // Non-initators : Upon reception of **add**(id_k)
$label_i^k := label_i^k \cup \{\mathrm{id}_k\}$;
$parent_i := id_k$;
$received\_add_i := received\_add_i \cup \{\mathrm{id}_k\}$;

4. // Non-initators : Upon reception of **search**(σ_k)
$\sigma_i := id_i \cup \sigma_k$;
forall $k \in \mathrm{Neig}_i \wedge \mathrm{numbered}_i^k =$ **false do**
begin
send add(id_i) to node n_i;
$label_i^k := label_i^k \cup \{\mathrm{id}_i\}$;
end

if $\exists k \in \mathrm{Neig}_i \wedge \mathrm{numbered}_i^k =$ **false then**
begin
send report)(σ_i) **to** $parent_i$**;**
return;
end

send search(σ_i) **for** k**, max**$label_i^k$ | $\forall$**k**$\in$**Neig**$_i \wedge$ **numbered**$_i^k$ **= false;**

$numbered_i^k$ **:= true;**

5. // **Initiators, Non-initiators : Upon reception of report**(σ_k)
σ_i **:=** σ_k**;** // **Overwrite**
if $parent_i \neq$**id**$_i$ **then**
send report(σ_i) **to** $parent_i$**;**
else
initiate maximal_clique_procedure;

6. // **Construct maximal clique cover set from** σ
$clique\_id_i$ **:=** id_i**;**
$maximal\_clique\_set_i := \emptyset$;

forall $k \in$**Neig**$_i \wedge \sigma^{-1}(i) < \sigma^{-1}(k)$ **do**
begin
$maximal\_clique\_set_i$ **:=** $id_k \cup$**maximal_clique_set**$_i$**;**
send join($maximal\_clique\_set_i$**) to node** n_k**;**

if $clique\_id_i < id_k$ **then**
$clique\_id_i$ **:=** id_k**;**
end

7. // **Upon reception of join(**$maximal\_clique\_set_k$**)**
if $|\, maximal\_clique\_set_k \,| > |\, maximal\_clique\_set_i \,|$ **then**
begin
forall k$\in$**maximal_clique_set**$_i$ **do**
send reject($maximal\_clique\_set_i$**) to node** n_k**;**

$maximal\_clique\_set_i$ **:=** $maximal\_clique\_set_k \cup$**{id**$_i$**};**
forall $j \in$**maximal_clique_set**$_k$ **do**
if $n_j \neq$**Neig**$_i$ **then**
begin
send reject($maximal\_clique\_set_k$**) to node** n_k**;**
return;
end

$maximal\_clique\_set_i$ **:=** $maximal\_clique\_set_k$**;**
$clique\_id_i$ **:=** **max**id_j $|\, \forall$**j**$\in$**maximal_clique_set**$_k$**;**
end
else
send reject(id_i**) to node** n_k**;**

8. // **Upon reception of reject(**$maximal\_clique\_set_k$**)**
if $maximal\_clique\_set_k = maximal\_clique\_set_i$ **then**
begin
// **Reset values**
$maximal\_clique\_set_i$ **:=** $\emptyset$**;**
clique_id$_i$ **:=** id_i**;**

forall $j \in$**maximal_clique_set**$_k$ **do**
send reject($maximal\_clique\_set_k$**) to node** n_j**;**

end

9. **// Calculate the overall network leader**
forall $k \in$**Neig**$_i$ **do**
begin
if $clique\_id_i < clique\_id_k$ **then**
$clique\_id_i$ **:=** $clique\_id_k$**;**
end

forall $k \in$**Neig**$_i$ **do**
if $clique\_id_i \neq$**clique_id**$_k$ **then**
send new_leader($clique\_id_i$**);**

10. **// Upon reception of new_leader(**id_k**)**
if $clique\_id_i < id_k$ **then**
$clique\_id_i$ **:=** id_k**;**

Claim.

Algorithm Elect_Leader elects a leader in an interval network with a best case messaging complexity $O(n)$ with worst case messaging complexity of $O(n^2)$.
Proof.

Steps a-e generate a perfect elimination scheme in a distributed

system. See proof [Golumbic]. In achieving this, a total of $2n-1$ messages are exchanged among the nodes as proved by the lemme below.

Steps f-h constructs the maximal clique sets with a best case messaging complexity of $\lceil \frac{n}{2} \rceil$.

Steps i-j exchange as many as $\lceil \frac{n}{2} \rceil$ **new_leader** messages for a node to announce its finding to all the nodes in its clique set. Hence a total of $O(n)$ best message complexity is achieved by the algorithm.

Lemma 3.1

A total of $3n-3$ messages are exchanged between nodes in steps 1-5 in the algorithm to achieve a perfect elimination scheme σ. Upon completion, all the nodes in the perfect scheme know the entire scheme for the network.
Proof.

Atmost n steps of processing takes place among the various nodes

in the system and the perfect elimination scheme σ is constructed. Each node initiates atmost one node in the process of searching **search** (the node which is un-numbered and which has the largest label) which adds to a total of $n-1$ messages.

add message is sent to all other nodes which are currently not involved in the search procedure. We can see from the sequence of steps that the number of add messages that are sent are $n-1$ (for all the edges in the network).

Atmost $n-1$ **report** messages are received for announcing the termination which adds to a total of $3n-3$ (or $3|\ E\ |$).

Claim.

For any node n_i already in the perfect scheme σ, the largest label of all the un-numbered vertices is adjacent to the node n_i. This node proceeds to initiate the search procedure for the largest label of the un-numbered adjacent.

Claim.

The algorithm calculates the set of maximal cliques with a best case message complexity of $\lceil \frac{n}{2} \rceil$ and worst case message complexity $\frac{n^2-n-4}{2}$ (for the case of a single-clique network

Lemma 3.2

For the node n_j at position $\sigma^{-1}(j)$, atmost $n-\sigma^{-1}(j)-1$ **reject** messages are propagated back.

Proof.

Given the perfect scheme σ, each of the node tries to concurrently find and

group with the maximum number of adjacent nodes possible to maximize the chance of constructing its clique set. In doing so, a node at position $\sigma^{-1}(j)$ (the position of id j in the scheme σ) could recieve $n-\sigma^{-1}(j)-1$ since its already grouped with atleast a single node to the right of this node in the scheme and which is adjacent.

Corollary 3.3

Upon completion of the steps f-h in the algorithm, each of the node in the perfect scheme σ is associated with a clique identified by an unique $clique\_id_i$

Claim.

Upon completion of steps i-j, all nodes have the identifier of the overall leader of the network

Lemma 3.4

A total of $\lceil \frac{n}{2} \rceil - 1$ **new_leader** messages are exchanged in the worst case.
Proof.

In the worst case, maximal clique sets are

pair-wise disjoint such that atleast 2 nodes are involved in a clique set and one of the nodes connect to the node of other clique set and so on. Hence, it takes $\frac{n}{2}$ level to propagate the overall leader of the network in the worst case.

References

[R.Gallager, R.G., Humlet, P.A. and Spira, P.M.] A Distributed Algorithm for Minimum Weight Spanning Trees, ACM Trans. Program. Syst., Vol. 5. pp 66-77 Jan,1983

[Fulkerson, D.R., Gross, O.A.] Incidence matrices with the consecutive 1's propert, Bull. Amer. Math. Soc. 70 [1964] Incidence matrices and interval graphs, Pacific J. Math.15 [1965]

[Gilmore, Paul C., and Hoffman, Alan J.] A characterization of comparability graphs and of interval graphs, Canada. J. Math. 16 [1964]

[M.C.Golumbic] Algorithmic Graph Theory and Perfect Graphs Academic Press Inc.

[E.Korach, S.Moran, S.Zaks] Tight Lower and Upper Bounds for some Distributed Algorithms for a Complete Network of Processors, Proc. of the Third Annual ACM symp. on Principles of Distributed Computing (1984) pp. 199-207

[D.Dolev, M.Klawe and M.Rodeh] An $O(n \log n)$ Unidirectional Distributed Algorithm for Extrema-Finding in a Circle, Journal of Algorithms, Vol. 3, Sep,1982

[G.L.Peterson] Efficient Algorithms for Elections in Meshes and Complete Networks, University of Rochester, Aug,1984

[Gerard Tel] Introduction to Distributed Algorithms Cambridge University Press.

[Yossi Matias and Yehuda Afek] Simple and Efficient Election Algorithms for Anonymous networks, Lecture Notes on Computer Science, Distributed Algorithms [1997]

[Hagit Attiya] Constructing Efficient Election Algorithms from Efficient Traversal Algorithms, Lecture Notes on Computer Science, Distributed Algorithms [1997]

[Gurdip Singh] Efficient leader election using sense of direction, Distributed Computing 10 [1997]