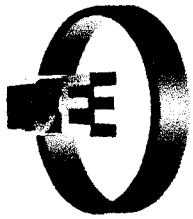
Núcleo de Computação Eletrônica



On Computing All Maximal Cliques Distributedly

Fábio Protti
Felipe M. G. França
Jayme Luiz Szwarcfiter

NCE - 01/97
abril

Relatório Técnico

Universidade Federal do Rio de Janeiro

On Computing All Maximal Cliques Distributedly*

Fábio Protti
Felipe M. G. França
Jayme Luiz Szwarcfiter
Universidade Federal do Rio de Janeiro, Brazil
email: {fabiop, felipe, jayme}@cos.ufrj.br

Abstract. A distributed algorithm is presented for generating all maximal cliques in a network graph, based on the sequential version of Tsukiyama et al. [TIAS77]. The time complexity of the proposed approach is restricted to the induced neighborhood of a node, and the communication complexity is $O(md)$ where m is the number of connections, and d is the maximum degree in the graph. Messages are $O(\log n)$ bits long, where n is the number of nodes (processors) in the system. As an application, a distributed algorithm for constructing the *clique graph* $k(G)$ from a given network graph G is developed within the scope of dynamic transformations of topologies.

1 Introduction

The generation of all possible configurations with a given property is a problem that occurs frequently in many distinct situations. In terms of distributed systems, the recognition of all subgraphs possessing special features within a network graph may simplify distributed algorithms developed for the network [CG90]. Among the different types of subgraphs, maximal cliques (completely connected subgraphs) play a major role. For instance, the analysis of maximal cliques provides a better understanding of the behavior of scheduling mechanisms [BG89] and the efficiency of mappings [FF95] for neighborhood-constrained systems.

In this paper, we propose a distributed algorithm for generating all maximal cliques of an arbitrary network graph. A previous approach to parallel distributed computation of all maximal cliques of a given graph was presented in [JM92]. The latter is based on the parallel version proposed in [DK88] and in order to compute all maximal cliques, divide-and-conquer strategy is used by dividing the node set into two disjoint subsets V_1, V_2, finding the maximal cliques in V_1, V_2 recursively, and joining these cliques via maximal bipartite complete subgraphs of the graph $G'(V_1 \cup V_2, E')$, where E' is the subset of edges with one endpoint in V_1 and other endpoint in V_2.

The present method is based on the concurrent execution of a local sequential algorithm [TIAS77] for all induced neighborhoods. The time complexity is restricted to the induced neighborhood of a node, and the communication complexity is $O(md)$ where m is the number of connections, and d is the maximum

* This work has been partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) Brazil.

degree in the graph. Messages are $O(\log n)$ bits long, where n is the number of nodes (processors) in the system.

This work is organized as follows. Section 2 describes the distributed processing environment. Section 3 reviews the sequential algorithm of Tsukiyama et al. [TIAS77]. Section 4 describes the present algorithm and analyzes its complexity. Section 5 contains the following application: a distributed algorithm for constructing the *clique graph* $K(G)$ from a given network graph G. The graph $K(G)$ has the set of maximal cliques of G as node-set, and there exists an edge (C, C') in $K(G)$ iff $C \cap C' \neq \emptyset$. Some remarks form the last section.

2 The Distributed Processing Model

Consider an arbitrary connected graph $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, where each node in V corresponds to a complete local processing environment containing a processor and sufficient local memory to perform computations. Notice that we assume implicitly a set of distinct identifications of nodes having a *total ordering*.

Two nodes i and j are *neighbors* if $(i, j) \in E$. The adjacency set $N(i) = \{j \in V | (i, j) \in E\}$ is the *neighborhood* of node i. The cardinality of $N(i)$ is the *degree* d_i of node i. If $X \subseteq V$, we define $G[X]$ as the *subgraph of G induced by X*: its node-set is X, and a connection $e = (i, j)$ in E belongs to the edge-set of $G[X]$ if and only if $i, j \in X$. For each node i, $G[N(i)]$ is the induced neighborhood of i.

A *clique* $C \subseteq V$ is a completely connected set of nodes. That is, if $C = \{i_1, i_2, \ldots, i_k\}$, then $(i_r, i_s) \in E$ for all $r, s \in \{1, 2, \ldots, k\}$, $r \neq s$. A clique C is *maximal* if, whenever we have $C \subseteq C'$ for a clique C' in V, then $C = C'$.

Messages can flow independently in both directions between neighboring nodes through the connection linking them. Each node knows its neighborhood, and maintains internally an input buffer for arriving messages.

We will assume that each node of G is able to execute the communication instruction $BROADN(msg)$ [BF88, BDH94]. Used in some massively parallel applications, $BROADN(msg)$ works simply as follows: when executing it, node i sends a message msg to all of its neighbors.

We will also assume that in the distributed model all nodes execute the same algorithm. Message transmission time is not taken into account, since the distributed algorithm is totally message-driven, not only inside each of its local steps, but also between the steps.

3 The Sequential Version

First, we review the sequential algorithm for generating all maximal independent sets developed in [TIAS77] (see also [L76], [PU59]), in its maximal clique version. The proofs of lemmas in this section can be found in [L76].

Let H be a connected graph with node-set $V(H) = \{1, 2, \ldots, p\}$. Let C_j denote the collection of all maximal cliques in $H[\{1, 2, \ldots, j\}]$. The idea is to compute C_{j+1} from C_j, eventually computing C_p.

Let $C \in C_j$. If $C \cap N(j+1) = C$, then $C \cup \{j+1\} \in C_{j+1}$. If $C \cap N(j+1) \neq C$, then $C \in C_{j+1}$ and $(C \cap N(j+1)) \cup \{j+1\}$ is a clique (not necessarily maximal). Let $C'_{j+1} = \{C' | C' = (C \cap N(j+1)) \cup \{j+1\}, C \in C_j\}$.

The observations above lead to the following lemma:

Lemma 1. $C_{j+1} \subseteq C_j \cup C'_{j+1}$.

Lemma 1 provides a way of generating algorithmically C_{j+1} from C_j. One point to be considered is how to avoid inclusion of nonmaximal sets in C_{j+1}, since $C_j \cup C'_{j+1}$ may contain sets of that sort. This is the subject of Lemma 2.

Lemma 2. *Let $C \in C_j$. Then, $C' = (C \cap N(j+1)) \cup \{j+1\} \in C_{j+1}$ if and only if for all $k < j, k \notin C$, we have $N(k) \cap C' \neq C'$.*

Another point is how to avoid inclusion of a maximal set which has already been included. To deal with this second question, we may regard C'_{j+1} as a multiset containing *duplicates*. That is, a given set $C' \in C'_{j+1}$ may be obtained from distinct sets $C^1, C^2 \in C_j$ such that $C' = (C^1 \cap N(j+1)) \cup \{j+1\} = (C^2 \cap N(j+1)) \cup \{j+1\}$. In order to avoid duplicates, we use the following rule: given a set $C \in C_j$, we include $C' = (C \cap N(j+1)) \cup \{j+1\}$ into C'_{j+1} *only if $C - N(j+1)$ is lexicographically smallest among all sets $C \in C_j$ for which $C \cap N(j+1)$ is the same* (all sets are implicitly organized as lists in increasing order). Lemma 3 tells us how to check this property for a given set.

Lemma 3. *A set $C \in C_j$ satisfies the lexicographic condition if and only if there is no node $k < j+1, k \notin C$, such that:*
a) k is adjacent to $X = C \cap N(j+1)$, that is, $N(k) \cap X = X$;
b) k is adjacent to all lower-numbered nodes in $C - N(j+1)$, that is, $N(k) \cap Y = Y$, where $Y = \{1, 2, \ldots, k-1\} \cap (C - N(j+1))$.

The algorithm below follows from Lemmas 1-3:

proc *Generate*(C_j, C_{j+1})

```
for C ∈ C_j do
    if C ∪ {j+1} is a clique
        then include C ∪ {j+1} in C_{j+1}
        else
            include C in C_{j+1};
            C' := (C ∩ N(j+1)) ∪ {j+1};
            if C' and C satisfy Lemmas 2 and 3 then include C' in C_{j+1};
end
```

The following routine computes C_p:

proc *Generate_maximal_cliques*(H, C_p)
% Input: connected graph H with node-set $\{1, \ldots, p\}$
% Output: C_p, the set of all maximal cliques of H

$C_1 := \{\{1\}\}$;
for $j := 1$ **to** $p - 1$ **do call** *Generate*(C_j, C_{j+1});
end

The tree of Figure 1 illustrates the computation. Nodes at level j correspond to the sets in C_j. The root of the tree is $\{1\}$, the only set in C_1. Each node has at least one son because for each $C \in C_j$, either C or $C \cup \{j+1\}$ is in C_{j+1}. The left son of a node $C \in C_j$, if it exists, corresponds to C. It exists only if $C \cap N(j+1) \neq C$, otherwise C would not be maximal in C_{j+1}. The right son exists only if the maximality and the lexicographic conditions hold. In this case, it corresponds to $(C \cap N(j+1)) \cup \{j+1\}$.

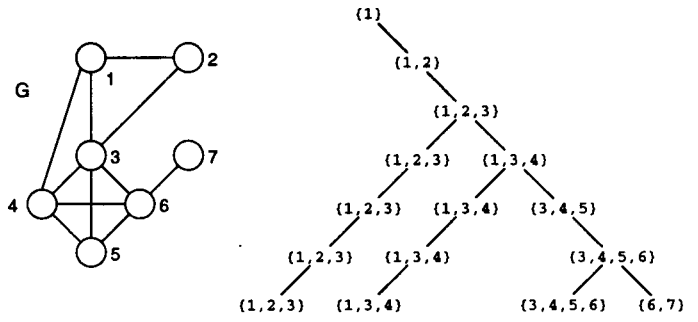


Fig. 1. Tree representing the generation of all maximal cliques for the graph G.

Let q be the number of edges and M be the number of maximal cliques in H. Both tests of Lemmas 2 and 3 can be performed for C' and C in $O(p+q)$ time. Therefore, we can compute C_{j+1} from C_j in $O((p+q)M)$ time. That is, C_p can be obtained in $O((p+q)p)M = O(qpM)$ time. The computation can be done by a depth-first search of the tree. If the leaves are to be output, space requirements are reduced to $O(p)$ cells, each one containing at most p nodes.

4 The Distributed Version

The distributed algorithm consists of two asynchronous steps. At the end, each node will have computed *all the maximal cliques it belongs to.*

4.1 Step 1: Constructing induced neighborhoods

The objective of this step is that all of the system's nodes may know their induced neighborhoods. Therefore, every node i starts by notifying all of its neighbors about its own adjacency relations by executing the following subroutine:

proc *Broadcast_neighborhood*(i)

for $j \in N(i)$ **do** $BROADN(adj(i,j))$;
$BROADN(broad\_end(i))$;
end

An $adj(i,j)$ message means that i and j are neighbors, and a $broad\_end(i)$ message warns the neighborhood of i to consider the broadcasting of the adjacency relations concluded.

Next, i is ready to process information that has just been received from its neighbors. Upon receiving a message $broad\_end(j)$ from every node $j \in N(i)$, the construction of $G[N(i)]$ is concluded.

proc *Construct_induced_neighborhood*(i)
% G_i is the induced neighborhood $G[N(i)]$ to be constructed

for $j \in N(i)$ **do** include edge (i,j) in G_i;
for an $adj(j,k)$ message arrived before a $broad\_end(j)$ message **do**
if $j,k \neq i$ **and** $j,k \in N(i)$ **then** include edge (j,k) in G_i;
end

Now, let m be the number of connections and d the maximum degree in the system.

Lemma 4. The first step takes $O(d_i^2)$ local computation time for node i, and its communication complexity is $O(md)$.

Proof. Executing $BROADN$ instructions and collecting $adj(j,k)$ messages to construct $G[N(i)]$ takes $O(d_i^2)$ time, therefore the first part follows. After all nodes have completed the first phase, the number of messages is $\sum_{(i,j)\in E} d_i + d_j + 2 = O(md)$. Notice that each message can be formed with $O(\log n)$ bits. □

4.2 Step 2: Computing all maximal cliques

Having constructed $G[N(i)]$, node i is able to compute internally and sequentially the maximal cliques it belongs to. The essence of the second step is the concurrent execution of the sequential algorithm we described in Section 3. Let C be a maximal clique containing i. Of course, $G[C-\{i\}] \subseteq G[N(i)]$. This suggests that node i must execute the sequential algorithm applied to its induced neighborhood.

proc *All_maximal_cliques*(i)

$H \leftarrow \emptyset$; $p \leftarrow d_i$;
map nodes of $G[N(i)]$ into $\{1,\ldots,d_i\}$ so that $f(j) < f(k)$ iff $j < k$;
for $(j,k) \in G[N(i)]$ **do** include edge $(f(j), f(k))$ in H;
call *Generate_maximal_cliques*(H, C_p); % local sequential algorithm
for $B \in C_p$ **do**
let C be the set of nodes in $\{f^{-1}(j) | j \in B\} \cup \{i\}$;

order C increasingly as a list;
store (or output) C;
end

In order to generate each clique as a list in increasing order of nodes, nodes of $G[N(i)]$ are mapped into $V(H) = \{1, 2, \ldots, d_i\}$ in such a way that relative positions between nodes be preserved. After generating a clique, i must itself be included in it.

Lemma 5. *Let m_i be the number of connections in $G[N(i)]$ and M_i the number of maximal cliques containing i. Then, the second step takes $O(m_i d_i M_i)$ local computation time for i, and no messages are sent.*

Proof. Consider the complexity of the algorithm in [TIAS77] when applied to $G[N(i)]$. □

4.3 Analysis of the algorithm

Asynchronicity is an interesting feature of the proposed algorithm, as each node can start independently the first step and enter the second step even though other nodes in the system may have not completed the first step yet. This facilitates considerably the development of actual distributed and parallel applications. Lemmas 4 and 5 lead to the following theorem, which synthetizes the properties of the algorithm.

Theorem 6. *The time complexity of the algorithm is $\max_{i \in V}\{O(m_i d_i M_i)\}$. The communication complexity is $O(md)$, with $O(\log n)$-bit long messages.*

The communication complexity of the algorithm in [JM92] is $O(M^2 n^2 \log n)$, where M is the number of maximal cliques and there is a condition on the messages, *assumed* to be $O(\log n)$ bits long. In [JM92], since there are transmission of messages containing *identifications of cliques*, message length becomes a function of M. Thus, it has been assumed that M is polinomial on n in order to ensure that message length is not higher than $O(\log n)$. In the present algorithm, there is no such restrictions: the communication mechanism does not depend on M, and message length is indeed limited by $O(\log n)$.

Notice that the time complexity of the proposed algorithm is limited to one of the induced neighborhoods of the system. On the other hand, the $O(Mn \log n)$ time complexity of [JM92] refers to message transmission time.

Our algorithm is time efficient, of course, for classes of systems in which the number of maximal cliques is polynomially bounded. The time performance is also good for topologies with constrained connectivity, i.e., when the maximum degree d is limited (a quite natural restriction for many actual systems).

The distributivity of the computation among the processors is weaker in systems with universal nodes (nodes u for which $N(u) = V - u$), since every node of this kind computes all maximal cliques. On the other hand, as the number of universal nodes increases, the number of maximal cliques tends to decrease (in the extreme case of a complete graph, there is one maximal clique only).

5 An application: clique graph construction

The subject of this section is within the scope of dynamic transformations of topologies according to a pre-established rule or *operation*. An operation $op : U \rightarrow U$ is a function over the set U of all graphs. Now, let N be a network graph on which a certain operation op will be applied. This means that we will run on N a distributed algorithm A, which by creating and removing nodes and connections, will modify the topology of N transforming it into a new system N' such that $N' = op(N)$. By running A on N', we generate N'', which by its turn corresponds to $op(N')$. That is, repeated executions of A correspond to iterated applications of op.

The operation we consider is the construction of the *clique graph* $K(G)$ from a given graph G. The node-set of $K(G)$ is the set of all maximal cliques in G, and there is an edge between two nodes of $K(G)$ whenever the corresponding cliques in G share a common node. Detailed information on clique graphs can be found in [P95].

The construction of $K(G)$ can be briefly described as follows. The nodes of the connected network graph $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, start running independently copies of the same algorithm. At the end, the system will be identified with $K(G)$, in the sense that new nodes correspond to maximal cliques of G, and new connections to node-sharing between cliques.

Before describing the algorithm, it is necessary to detail an additional set of instructions for communication between nodes (see [BF88, BDH94]). Assume each of the following instructrions is being executed by node i.

- $SEND(j, msg)$: node i sends a message msg destined to node j. If i and j are neighbors the communication is direct. Otherwise, the instruction needs to be re-transmitted by other nodes. This means that routing information indicating the next node to send arriving messages is required in general. This mechanism is independent of internal computations in the nodes, i.e., a node only processes messages sent to it. If n is the number of nodes in the system, an instruction $SEND$ may require $n-1$ transmissions, and this fact must be taken into account when calculating the communication complexity of a distributed algorithm. If j does not exist, msg is discarded after visiting all of the nodes.
- $CREATE(id)$: this is a high-level instruction where node i creates a new neighbor and assigns to it the identification id.
- $LINK_1(k, j)$: this is another high-level instruction where node i creates a connection between nodes k and j belonging to its neighborhood.
- $LINK_2(j)$: node$i$ creates a connection between itself and j. If i and j are already neighbors, this instruction has no effect.
- $UNLINK(j)$: node i removes the connection between itself and j.

The construction of the clique graph has three steps.

5.1 Step 1: local computing of maximal cliques

In this step each node i simply executes the algorithm of the previous section for generating all maximal cliques it belongs to. The only difference is that i

discards every clique C whose first node in the list is different from i (recall that the cliques are generated as lists in increasing order of nodes). This means that the set of all maximal cliques of G becomes *partitioned* among the nodes. Notice that some nodes will store no cliques.

Lemma 7. Step 1 takes $O(m_i d_i M_i)$ time for node i. The communication complexity of this step is $O(md)$, with $O(\log n)$-bit long messages.

Proof. See Theorem 6. □

5.2 Step 2: generating clusters

Once i has completed Step 1, it starts immediately Step 2 by generating a *cluster*, a clique formed by new nodes where each one represents a clique C stored in i at the end of Step 1 and receives C as its identification. See Figure 2.

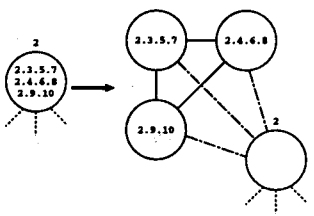


Fig. 2. Generation of a generic cluster formed by 3 nodes.

Observe that each cluster must be completely connected, since it consists of nodes identifying pairwise non-disjoint cliques of G. Observe also that new identifications are formed for ordered lists of nodes in G. This means that the set of identifications of $K(G)$ will also have a total (lexicographic) ordering. The processing of Step 2 can be described in the following way:

proc *Step_2*(i)

  for a clique C stored in i **do** $CREATE(C)$;
  for distinct cliques C, C' stored in i **do** $LINK_1(C, C')$;
end

For the sake of simplicity, from now on we will refer indistinctly to the *node* C in $K(G)$ and the *clique* C in G. Figure 3a shows the system G in Figure 1 after all nodes have completed Step 2.

Lemma 8. Step 2 takes $O(M_i^2)$ time for node i. No messages are sent in this step.

Proof. The second instruction **for** consists of M_i^2 iterations. □

5.3 Step 3: making links between clusters

Let us call *leaders* the *original nodes* of the system G. Notice that only leaders run the algorithm.

The processing in Step 3 is the following:

proc *Step_3*(i)

% notifying other leaders - - - - -
for a clique C belonging to the cluster of i **do**
let j be the highest node in C;
for $k = i + 1$ **to** j **do** $SEND(k, clique(C))$;

% synchronization - - - - -
if $i = 1$
then $SEND(i + 1, end\_notify(i))$
else
collect all *clique*( ) messages arriving before a $end\_notify(i - 1)$ message;
$SEND(i + 1, end\_notify(i))$;

% making links - - - - -
for a $clique(C)$ message collected **do**
% messages referring only to $SEND(i, clique(C))$ instructions
for a clique C' belonging to the cluster of i **do**
if $C' \cap C \neq \emptyset$ **then**
$LINK_2(C)$;
$LINK_1(C', C)$;
$UNLINK(C)$;

% removing - - - - -
for a leader $j \in N(i)$ **do** $UNLINK(j)$;
for a clique $C \in N(i)$ **do** $UNLINK(C)$;
end

Node i starts notifying other leaders about the nodes belonging to the cluster it leads. A message $clique(C)$ means that C is a clique. For each node C belonging to the cluster of i, all leaders whose clusters may contain some node C' such that $C \cap C' \neq \emptyset$ are notified. Let max be the highest node in the union of the cliques i leads. Notice that there is no use notifying leaders with higher identifications than max, since they cannot lead a clique C' such that $C \cap C' \neq \emptyset$. Also, no leader k with identification smaller than i is notified, since k will certainly notify i in case when there exists a clique C in the cluster of k and a clique C' in the cluster of i such that $C' \cap C \neq \emptyset$.

Next, a synchronization point is needed: each leader, in order to perform the next stages ('making links' and 'removing'), must be sure that no more *clique*( ) messages will be received, that is, there are no more *clique*( ) messages to be processed.

Now let us deal with the matter of linking the nodes of $K(G)$. Every *clique*( ) message is (re-)transmitted between leaders only. Each leader i verifies for each

message *clique*(C) received whether there are cliques in its cluster intercepting C. For each clique C' with this property, an instruction $LINK_1(C', C)$ is executed. Figure 3b shows the system after all the leaders have executed this linking process.

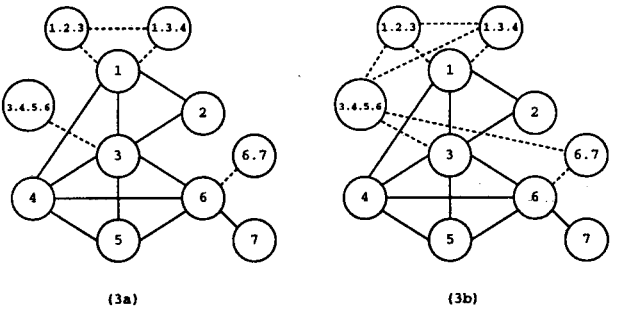


Fig. 3. The system after generating clusters (3a) and after making links between nodes in distinct clusters (3b).

Finally, notice that by removing the leaders we obtain a system corresponding exactly to the clique graph $K(G)$. Since $K(G)$ is a connected graph whenever G is also connected, the resulting system after removing the leaders is connected.

Lemma 9. *Step_3 takes* $O(d_i M_i \sum_{j<i} M_j) = O(d_i M_i M)$ *time for node* i. *The communication complexity of this step is* $O(ndM)$, *with* $O(d \log n)$*-bit long messages.*

Proof. Notifying other leaders requires $O(nM_i)$ time, and synchronization requires $O(\sum_{j<i} M_j)$ time. With respect to making the links, recall that i does not receive any message *clique*( ) from leaders with higher identifications than i. Since the internal **for** requires $O(d_i M_i)$ time, links are made in $O(d_i M_i \sum_{j<i} M_j)$ time. Removing requires simply $O(d_i)$ time, therefore the time complexity of the third phase follows. The number of messages corresponds to the number of $SEND$ instructions executed overall. Each *clique*( ) message is retransmitted at most n times, thus the communication complexity of this phase is $O(\sum_{i=1}^{n} n d_i M_i) = O(ndM)$. Since each clique has $O(d)$ nodes, message length is $O(d \log n)$. □

By Lemmas 7,8 and 9, we obtain the complexity for constructing $K(G)$. Compare it with the sequential-time complexity, which is $O(mnM + dM^2)$.

Theorem 10. *The time complexity of the clique graph construction algorithm is* $\max_{i \in V}\{O(m_i d_i M_i + d_i M_i M)\}$, *and its communication complexity is* $O(ndM)$, *with* $O(d \log n)$*-bit long messages.*

It remains to prove the correctness of the construction and that no redundant linking instructions are executed.

Theorem 11. Let C and C' be two nodes of $K(G)$ created in Step 2.
a) if a linking instruction between C and C' is executed, then $C \cap C' \neq \emptyset$;
b) if $C \cap C' \neq \emptyset$, then exactly one linking instruction between C and C' is executed.

Proof. The proof of *a)* is straightforward, since any $LINK$ instruction between C and C' is only executed either when constructing a cluster in Step 2 or after an intersection test in Step 3. In both cases, $C \cap C' \neq \emptyset$. For the proof of *b)*, let C and C' be two cliques of $K(G)$ such that $C \cap C' \neq \emptyset$ and C is lexicographically smaller than C'. Let k be the smallest node in $C \cap C'$. There are two cases:

i) k is the smallest node in C. In this case, k is also the smallest node in C'. Therefore, C and C' belong to the same cluster generated in Step 2. By construction of the clusters, exactly one linking instruction between C and C' is executed.

ii) k is not the smallest node in C. Let l and m be the smallest nodes in C and C', respectively. Of course, $l < m \leq k$. Observe that leader l executes a $SEND(m, clique(C))$ instruction, and leader m executes a $LINK(C, C')$ instruction, since C' is a clique in the cluster of m and $C \cap C' \neq \emptyset$. Moreover, every instruction of the form $LINK(*, C')$ is executed only inside node m. Since $SEND(m, clique(C))$ is executed by l exactly once, $LINK(C, C')$ is executed exactly once. □

6 Conclusions

A simple distributed algorithm for generating all maximal cliques of any arbitrary network graph has been presented. As an application, a method for constructing clique graphs distributedly has also been described.

References

[BDH94] V. C. BARBOSA, L. M. de A. DRUMMOND, and A. L. H. HELLMUT, From distributed algorithms to Occam programs by successive refinements, The J. of Systems and Software 26 (1994), pp. 257-272.

[BF88] V. C. BARBOSA and F. M. G. FRANÇA, Specification of a communication virtual processor for parallel processing systems, in Proc. of Euromicro-88 (1988), pp. 511-518.

[BG89] V. C. BARBOSA and E. GAFNI, Concurrency in heavily loaded neighborhood-constrained systems, ACM Transactions on Programming Languages and Systems 11 (1989), pp. 562-584.

[BS96] V. C. BARBOSA and J. L. SZWARCFITER, Generating all acyclic orientations of an undirected graph, Technical Report ES-405/96, COPPE/Federal University of Rio de Janeiro.

[CG90] I. CIDON and I. S. GOPAL, Dynamic Detection of Subgraphs in Computer Networks, Algorithmica 5 (1990), pp. 277-294.

[DK88] E. DAHLHAUS and M. KARPINSKI, A fast parallel algorithm for computing all maximal cliques in a graph and related problems, Proc. of the first Scandinavian Workshop on Algorithm Theory (1988),pp. 139-144.

[FF95] F. M. G. FRANÇA and L. FARIA, Optimal mapping of neighborhood-constrained systems, in A. Ferreira and J. Rolim eds., Lecture Notes in Computer Science 980, pp. 165-170.

[JM92] E. JENNINGS and L. MOTYCKOVA, A distributed algorithm for finding all maximal cliques in a network graph, in I. Simon, ed., Lecture Notes in Computer Science 583, pp. 281-293, 1992.

[JYP88] D. S. JOHNSON, M. YANNAKAKIS, and C. H. PAPADIMITRIOU, On generating all maximal independent sets, Information Processing Letters 27 (1988), pp. 119-123

[L76] E. L. LAWLER, Graphical algorithms and their complexity, Mathematical Centre Tracts 81 (1976), Foundations of Computer Science II Part I, Mathematisch Centrum, Amsterdam, pp. 3-32.

[PU59] M. C. PAUL and S. H. UNGER, Minimizing the number of states in incompletely specified sequential functions, IRE Trans. Electr. Computers EC-8 (1959), pp. 356-357.

[P95] E. PRISNER, Graph Dynamics, Pitman Research Notes in Mathematics Series 338 (1995), Longman.

[TIAS77] S. TSUKIYAMA, M. IDE, H. ARUJOSHI and H. OZAKI, A new algorithm for generating all the maximal independent sets, SIAM J. Computing 6 (1977), pp. 505-517.

This article was processed using the LaTeX macro package with LLNCS style