A Formal Approach for Distributed Computing of Maximal Cliques in Dynamic Networks

Faten Fakhfakh[1], Mohamed Tounsi[1], Mohamed Mosbah[2], and Ahmed Hadj Kacem[1]
[1] ReDCAD Laboratory, University of Sfax, FSEGS, Tunisia
[2] LaBRI Laboratory, University of Bordeaux, France
faten.fakhfakh@redcad.org, mohamed.tounsi@redcad.org,
ahmed.hadjkacem@fsegs.rnu.tn, mohamed.mosbah@labri.fr

***Abstract*—The aim of this work is to propose a distributed algorithm, encoded by the local computations model, for computing maximal cliques in dynamic networks. This model provides an abstraction which simplifies the design and the proof of distributed algorithms. To guarantee the correctness of our algorithm, we use the Event-B formal method, which supports a refinement based incremental development using the RODIN platform.**

***Keywords*—Distributed algorithm, Local computations, Dynamic networks, Maximal cliques , Event-B method.**

I. Introduction

A. Motivation

Computing maximal cliques is a challenging problem in computer science which has found applications in several fields such as robotics, bioinformatics, etc. Many efforts have been dedicated to solve the problem of detecting maximal cliques. Most of the proposed approaches have used centralized algorithms and only few of them [6] [2] are based on distributed algorithms [7]. In the context of dynamic networks, the nodes are dynamically connected in an arbitrary manner without any established infrastructure or centralized administration. So, the topology of the network may change rapidly and unpredictably. Considering the complexity of distributed algorithms and the highly dynamic behavior, it is interesting to ensure the correctness of these algorithms to give us confidence that distributed systems perform as designed and do not behave harmfully.

According to our study, we notice that the majority of the existing works [2] [8] [6] for detecting maximal cliques rely on simulation to evaluate the performance of these solutions. Some works [2] [6] have proved the correctness of their algorithms based on formal proofs. Nevertheless, the proofs which have been presented are done manually. Also, these proofs are long and tedious specially in the case of complex algorithms and a minor error can have serious consequences on the system operation. To the best of our knowledge, only the solution of Xu et al. [8] dealt with dynamic networks. However, it has an exponential complexity.

B. Contribution

We propose in this paper a distributed algorithm for computing maximal cliques in dynamic networks inspired by the work of Luo et al. [6] which has a linear complexity. Our algorithm is based on the local computations model [5] that provides an abstraction for the design of distributed algorithms. To model dynamic networks, we use the evolving graph model [3] which consists in recording the evolution of the network topology as a discrete sequence of static graphs. Each static graph represents a snapshot of the dynamic network at a given date. In order to prove the correctness of the proposed algorithm, we use a formal method which offers a real help for expressing correctness with respect to safety properties in the design of distributed algorithms. The *correct-by-construction* approach [4] offers a simple way to specify and prove algorithms. It consists in developing distributed algorithms following a top/down approach controlled by the refinement of models. This process allows to simplify the proofs and validate the integration of requirements. The Event-B modeling language [1] can support this methodological proposal by suggesting proof based-guidelines.

DOI reference number: 10.18293/SEKE2018-109

C. Organization of the paper

The remainder of this paper is structured as follows: Section II introduces our proposed algorithm. In Section III, we specify this algorithm with the Event-B method. Finally, the last section concludes and provides insights for future work.

II. Algorithm for computing maximal cliques in dynamic networks : CMCDN

A network can be modeled as a simple and undirected graph *g=(V,E)* where *V* is the set of nodes and *E* is the set of edges. In this work, we suppose that every node in the graph knows its neighbours. A *"clique"* is a fully connected (or complete) subgraph of the graph *g* and a *"maximal clique"* is a clique that is not a subset of any other clique in the same graph. The proposed algorithm may be encoded by the graph relabeling system $R = (L, I', P)$. For a given node x, $L = \{State, Clique, N, Event\}$, $I' = \{Init, \varnothing, N(x), False\}$, and $P = \{R1, R2, R3, R4, R5, R6\}$. *State* and *Clique* are two functions. The node x has the following four labels:

- ***State(x)*** $\in$ {***Init, C, I, W, A***} is the state of the node x. It can take one of these labels: i) ***Init*** **:** the node x is in the initial state, ii) ***C*** **:** we call the node which detects a maximal clique *"the center"* of this clique, iii) ***I*** **:** the node x belongs to a maximal clique and it is different to the center of this clique, iv) ***W*** **:** the node x is in the waiting state. It has not been assigned to a maximal clique yet, v) ***A*** **:** the node x does not belong to any maximal clique. It is called *"isolated node"*.
- ***N(x)*** **:** It stores the node x and all its neighbours. The set $N(x)$ will be updated when an incident edge is removed.

• ***Clique(x)*** : The node x belongs to the maximal clique "$Clique(x)$". Initially, each node x of the graph has $Clique(x) = \varnothing$.
• ***Event(x)*** $\in$ {***False, Alert***} : It determines whether the node x belongs to an edge that has undergone a topological change or not. Initially, all nodes are in the state *"False"*. If an edge $u \mapsto x$ of a maximal clique has disappeared, then $Event(u)$ and $Event(x)$ take the state "$Alert$" to transmit a reconstruction order of the concerned clique.

I' is the set of initial labels and (***R1***, ***R2***, ***R3***, $R4$, $R5$ and $R6$) are the relabeling rules. For each node $x \in V$, we define $S(x) = \bigcap_{k \in N(x)} N(k)$ the set of nodes that stores the intersection of all $N(k)$ ($k \in N(x)$). We note $R(x)$ all the nodes of $S(x)$ which have not been assigned to maximal cliques yet. Therefore, these nodes are labeled *"Init"* or *"W"*.

To handle the disappearance of an edge during the construction of maximal cliques in a graph, we distinguish two cases :
Case 1 : Deleting an edge of the graph (an edge that does not belong to any maximal clique) has no effect on the maximal cliques already detected. This case requires applying the rule $R4$ (see Section II-D).
Case 2 : Deleting an edge belonging to a maximal clique can cause the destruction of this clique or the possibility of creating a new maximal clique.

A. The first rule: R1

The first rule aims to construct a maximal clique in the graph. Let x be a node in the initial state and $R(x)$ including at least three nodes. As a result, the nodes of $R(x)$ form a maximal clique having x as center. Formally, the rule ***R1*** is written as follows:
Precondition :

- $State(x) = Init$
- $card(R(x)) \geqslant 3$

Relabeling :

- $State(x) := C$ *and* $Clique(x) := R(x)$
- $\forall a \cdot a \in R(x) \setminus \{x\} \implies State(a) := I$ *and* $Clique(a) := R(x)$

B. The second rule: R2

The goal of the rule $R2$ is to attribute to a node that is unable to form a maximal clique the waiting state. This rule requires the presence of a node x in the initial state and $R(x)$ containing less than three nodes. Formally, the rule ***R2*** is written as follows:
Precondition :

- $State(x) = Init$
- $card(R(x)) < 3$
- $\exists k \cdot k \in N(x) \setminus \{x\}$ *et* $State(k) = Init$

Relabeling :

- $State(x) := W$

C. The third rule: R3

The purpose of the rule $R3$ is to identify an isolated node which does not belong to any maximal clique. Let x be a node in the initial or the waiting state and all the neighbouring nodes of x are not in the initial state. By applying $R3$, the label of the node x becomes *"A"* and $Clique(x)$ contains only the node x. Formally, the rule $R3$ is written as follows:
Precondition :

- $State(x) \in \{Init, W\}$
- $\forall a \cdot a \mapsto x \in g \implies State(a) \neq Init$

Relabeling :

- $State(x) := A$ *and* $Clique(x) := \{x\}$

D. The fourth rule: R4

The purpose of this rule is to express the modification made when deleting an edge of the graph. It requires the presence of an edge ($u \mapsto v$) that does not belong to any detected clique. The application of the rule ***R4*** causes the deletion of the edge $u \mapsto v$ from the graph as well as the updating of the sets $N(u)$ and $N(v)$. Formally, the rule ***R4*** is written as follows:
Precondition :

- $u \mapsto v \in g$
- $\neg$ (($Clique(u)$ = $Clique(v)$) and ($u \in Clique(u)$ and $v \in Clique(u)$))
- $Event(v) = False$ and $Event(u) = False$

Relabeling :

- $g := g \setminus \{u \mapsto v\}$
- $N(u) := N(u) \setminus \{v\}$ and $N(v) := N(v) \setminus \{u\}$

E. The fifth rule: R5

The role of this rule is to reflect the preliminary influence of the deletion of an edge belonging to a maximal clique. Let u and v be two nodes of the same maximal clique. While applying the rule $R5$, the edge $u \mapsto v$ disappears and $Event(u)$ and $Event(v)$ take the state "$Alert$". In addition, the sets $N(u)$ and $N(v)$ are updated. Formally, this rule is written as follows:
Precondition :

- $u \mapsto v \in g$
- $Clique(v) = Clique(u)$
- $State(v) \in \{C, I\}$ and $State(u) \in \{C, I\}$
- $Event(v) = False$ and $Event(u) = False$

Relabeling :

- $g := g \setminus \{u \mapsto v\}$
- $N(u) := N(u) \setminus \{v\}$ and $N(v) := N(v) \setminus \{u\}$
- $Event(v) := Alert$ and $Event(u) := Alert$

F. The sixth rule: R6

The rule $R6$ is used to reset the nodes states of a maximal clique. It requires the presence of a node (noted u) belonging to the maximal clique $\{u, a, ..., v\}$ and having "$Event(u) = Alert$".

• If there is a neighbour node of u (noted h) having the state A, then it resets the initial state. In fact, it can be a member of a maximal clique after resetting the states of nodes of the clique $\{u, a, ..., v\}$.

• If a node (noted b) which belongs to another maximal clique is the neighbour of the node u and can form a maximal clique (after destroying the clique $\{u, a, ..., v\}$) with the node u and other neighbours, then the node b takes the state $Alert$. Indeed, it must reset the states of the clique nodes by applying again the rule $R6$, to form a larger maximal clique. Formally, the rule $R6$ is written as follows:

Precondition :

- $Clique(u) = \{u, a, ..., v\}$ and $card(Clique(u)) \geq 3$
- $Event(u) = Alert$ and $(\forall k \cdot u \mapsto k \in g$ and $k \in Clique(u) \implies Event(k) = False)$

Relabeling :

- $\forall k \cdot k \in Clique(u) \implies State(k) := Init$
- $\forall k \cdot k \in Clique(u) \implies Clique(k) := \varnothing$
- $\forall k \cdot k \in Clique(u) \implies Event(k) := False$
- $\exists h \cdot \begin{pmatrix} u \mapsto h \in g \wedge State(h) = A \\ \wedge Clique(h) = \{h\} \end{pmatrix} \implies State(h) := Init \wedge Clique(h) := \varnothing$
- $\exists b \cdot \begin{pmatrix} u \mapsto b \in g \wedge card(Clique(b)) \geqslant 3 \\ \wedge Clique(b) \neq Clique(u) \\ \wedge (\forall r \cdot r \in Clique(b) \implies u \mapsto r \in g) \end{pmatrix} \implies Event(b) := Alert$

The propose rules of the algorithm CMCDN are applied asynchronously and non-deterministically until no rule is applicable. This means that many different runs are possible. In the final configuration, every node (x) has $State(x) \in \{C, I, A\}$, $Clique(x) \neq \varnothing$ and $Event(x) = False$:

- If x is an isolated node, it will have $State(x) = A$ and $Clique(x) = \{x\}$.
- If x belongs to a maximal clique, it will have $State(x) \in \{C, I\}$ and $card(Clique(x)) \geq 3$.

III. Formal specification of the algorithm CMCDN using the Event-B method

We introduce in this section our Event-B formal model of the algorithm CMCDN. It is based on four abstraction levels as shown in Fig. 1. The first machine **M0** abstractly specifies the goal of the proposed algorithm which aims to compute maximal cliques problem in dynamic networks. It utilizes properties defined in the context "Graph". The second machine **M1** refines M0. It contains some events which specify how nodes are making a choice to detect maximal cliques. Moreover, it globally specifies the consequences of deleting an edge. The machine **M2** refines M1. It provides more details to detect the set of nodes of each maximal clique and maintain the set of the detected cliques when deleting an edge. Finally, the machine **M3** refines M2 and sees the context "Labels". It includes a set of events corresponding to the six relabeling rules.

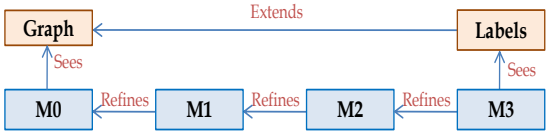

Fig. 1: The refinement strategy of the algorithm CMCDN

A. Formal specification of the contexts

1) The context "Graph" : This context specifies static properties of the network. Formally, a graph is modeled by a set of nodes called V. In our work, we suppose that a dynamic graph is composed of stable nodes. So, we define V in the context as an abstract set. We specify that the number of nodes in the network is finite $(axm1 : finite(V))$. Also, we introduce a constant, called "tn", that represents the final system date $(axm2 : tn \in \mathbb{N}1)$.

2) The context "Labels" : This context extends the context "Graph" by adding the labels of nodes to our model. Indeed, we introduce two sets called *"labels"* and "$event\_labels$". *"labels"* contains the labels : *Init*, *C*, *I*, *W* and *A* (*axm1* and *axm2*). The label allows nodes to perform an elementary step of computation according to some relabeling rules.
axm1 : $partition(labels, \{Init\}, \{C\}, \{I\}, \{W\}, \{A\})$
"$event\_labels$" includes the labels $Alert$ and $False$.
axm2 : $partition(event\_labels, Alert, False)$

B. Formal specification of the machines

1) The first level : The first machine M0 specifies the goal of the distributed algorithm, without describing the process of computing the solution (see Listing 1).

Listing 1: Machine M0 invariants

$inv1$	$g \in 0 \mathinner{..} t \rightarrow \mathbb{P}(V \times V)$
$inv2$	$\forall ti \cdot ti \in dom(g) \Rightarrow g(ti) = (g(ti))^{-1}$
$inv3$	$\forall ti \cdot ti \in dom(g) \Rightarrow (V \lhd id) \cap (g(ti)) = \varnothing$
$inv4$	$t \in \mathbb{N} \wedge t \leq tn$
$inv5$	$change \in \{0, 1\}$
$inv6$	$t \geq 1 \wedge change = 0 \Rightarrow g(t) = g(t-1)$
$inv7$	$cliques = \{V1, g1 \cdot V1 \subseteq V \wedge finite(V1) \wedge card(V1) \geq 3 \wedge g1 \subseteq g(t) \wedge g1 = (V1 \times V1) \setminus (V \lhd id) \wedge g1 = g1^{-1} \mid V1\}$
$inv8$	$combination \subseteq \mathbb{P}(cliques)$
$inv9$	$combination = \{X, x1, x2 \cdot X \subseteq cliques \wedge x1 \in X \wedge x2 \in X \wedge x1 \neq x2 \wedge x1 \cap x2 = \varnothing \wedge (\forall Y, a \cdot Y \in X \wedge a \in cliques \Rightarrow Y \cap a = \varnothing) \mid X\}$
$inv10$	$solution \in 0 \mathinner{..} t \rightarrow \mathbb{P}(\mathbb{P}(V))$
$inv11$	$final \in \{0, 1\}$

A network can be modeled as a simple and undirected graph g. It is defined as a function to be the graph at the current date t ($inv1$ and $inv4$). An undirected graph means that there is no distinction between two nodes associated with each edge ($inv2$). A graph is simple if it has zero or one edge between any two nodes and no edge starts and ends at the same node ($inv3$). Moreover, we introduce a variable called *"change"* (*inv5*). If one topological event has been produced, *"change"* is equal to "1", otherwise *"change"* is equal to "0". In the invariant *inv6*, we indicate that if the date t is strictly greater than "0" and *change* is equal to "0", the graph does not undergo any topological event. We define *"cliques"* as the set of all possible cliques in the graph g ($inv7$). Any pair of nodes can never form a clique, then all cliques contain

at least three nodes. We add *inv8* and *inv9* to specify all possible combinations of maximal and disjoint cliques in the graph *g* called "$combination$". Also, we introduce the variable *"solution"* which contains the result of the algorithm at the date t ($inv10$). The variable "$final$" allows to check if the $Oneshot$ event (will be explained later) has been triggered. The machine M0 includes three events :

• **Event *"Oneshot"*** (see Listing 2) : It specifies the result of the algorithm in one step. The analogy of someone closing and opening their eyes. In a given graph, there are many combinations of maximal cliques. The event *Oneshot* assigns in a non-deterministic way an element from *"combination"* to the variable *"solution"* at the date t.

Listing 2: Event *Oneshot*, in M0

EVENT $Oneshot$
ANY c
WHERE
$grd1$: $solution(t) = \varnothing$
$grd2$: $c \in combination$
$grd3$: $final = 0 \land t \neq tn$
THEN
$act1$: $solution(t) := c$
$act2$: $final := 1$
END

• **Event *"Remove_Edge"*** (see Listing 3) : An edge has been removed at the date *t* if it is present at the date "*t*" (*grd1*) and "*t*" is different from the final date *"tn"* (*grd2*). Then, we update the graph *g(t)* and the sets "$cliques$" and "$combination$". Also, the variable *"change"* takes the value "1" (*act2*).

Listing 3: Event $Remove\_Edge$, in M0

EVENT $Remove\_Edge$
ANY x, y
WHERE
$grd1$: $x \mapsto y \in g(t)$
$grd2$: $final = 0 \land t \neq tn$
THEN
$act1$: $g(t) := g(t) \setminus \{x \mapsto y, y \mapsto x\}$
$act2$: $change := 1$
$act3$: $cliques, combination : |cliques' = \{V1, g1 \cdot V1 \subseteq V \land$
$finite(V1) \land card(V1) \geq 3 \land g1 \subseteq (g(t) \setminus \{x \mapsto y, y \mapsto x\}) \land$
$g1 = (V1 \times V1) \setminus (V \lhd id) \land g1 = g1^{-1} | V1\}$
$\land combination' = \{X, x1, x2 \cdot X \subseteq cliques' \land x1 \in X \land$
$x2 \in X \land x1 \neq x2 \land x1 \cap x2 = \varnothing \land$
$(\forall Y, a \cdot Y \in X \land a \in cliques' \Rightarrow Y \cap a = \varnothing) | X\}$
END

• **Event *"Increment_Time"*** (see Listing 4) : This event can be activated if the current date *t* is strictly lower than the final system date *tn* ($grd1$), the result of the algorithm is verified ($grd3$) and at least one topological event is performed in the network ($grd2$). In the action component of this event, we increment the time to *t +1* ($act1$) and we set the graph at the date *t+1* to the graph *g(t)* (*act2*). In addition, we reset the variables *change*, $final$ and $solution$. Therefore, we have no topological change at the date *t+1*.

Listing 4: Event $Increment\_Time$, in M0

EVENT $Increment\_Time$
WHERE
$grd1$: $t < tn$
$grd2$: $change = 1$
$grd3$: $final = 1$
THEN
$act1$: $t := t + 1$
$act2$: $g(t + 1) := g(t)$
$act3$: $solution(t + 1) := \varnothing$
$act4$: $change := 0$
$act5$: $final := 0$
END

2) The second level : In the machine M1, we start by introducing details to calculate globally the maximal cliques. To do so, we add some variables to the invariant component as shown in Listing 5:

∘ "$cliques\_in$" contains the nodes which belong to the detected cliques ($inv1$).
∘ "$cliques\_out$" defines the set of nodes which do not belong to these cliques ($inv2$).
∘ "$cliques\_new$" is the set of nodes of the detected maximal and disjoint cliques ($inv5$ and $inv6$).
∘ "$adjacents$" gives the set of adjacent nodes to each node at the current date t ($inv9$ and $inv10$).
∘ "$nodes\_alert$" contains the nodes of the clique edge that has undergone a topological event ($inv11$).

Initially, $cliques\_out$ contains all the graph nodes *"V"*, whereas $cliques\_in$ and $cliques\_new$ are empty.

Listing 5: Machine M1 invariants

$inv1$: $cliques\_in \subseteq V$
$inv2$: $cliques\_out \subseteq V$
$inv3$: $cliques\_in \cap cliques\_out = \varnothing$
$inv4$: $cliques\_in \cup cliques\_out = V$
$inv5$: $cliques\_new \subseteq \mathbb{P}(cliques\_in)$
$inv6$: $\forall a, b \cdot a \in cliques\_new \land b \in cliques\_new \land a \neq b \Rightarrow a \cap b = \varnothing$
$inv7$: $\forall x \cdot x \in cliques\_new \land finite(x) \Rightarrow card(x) \geq 3$
$inv8$: $\forall x \cdot x \in cliques\_new \Rightarrow x \in cliques$
$inv9$: $adjacents \in V \times (0 \,..\, t) \rightarrow \mathbb{P}(V)$
$inv10$: $\forall ti, x \cdot ti \in (0 \,..\, t) \land x \in V \Rightarrow adjacents(x \mapsto ti) =$
$\{y \cdot x \mapsto y \in g(ti) \lor y \mapsto x \in g(ti) | y\}$
$inv11$: $nodes\_alert \subseteq V$
$inv12$: $\forall x \cdot x \in cliques\_out \Rightarrow x \notin nodes\_alert$

The definition of these variables requires the addition of new properties:
(*inv3*) The nodes of $cliques\_in$ are different from those of $cliques\_out$.
(*inv4*) The total of these nodes is equal to the set of nodes *V*.
(*inv7*) All the detected cliques contain at least three nodes.
(*inv8*) The set of the detected cliques is a subset of "$cliques$".
(*inv11*) The nodes of $cliques\_out$ can not belong to the set $nodes\_alert$.

At this level, we refine the events defined in M0 and we add some new events:

• **Event *"Clique+"* :** This event aims to compute maximal cliques in the graph *g* (see Listing 6). It can be activated if we have a ball *"B"* including at least three nodes ($grd3$ and $grd6$). We note *"x"* the center of the ball *B* ($grd2$ and $grd3$). All the nodes of *B* are connected ($grd4$) and belong to $cliques\_out$ ($grd1$). Using the *grd5*, we express that the ball *B* can not be extended by one or more adjacent nodes. At every computation step, the nodes of the detected ball are eventually added to the set $cliques\_in$ ($act2$) and removed from $cliques\_out$ ($act1$). Moreover, we add the set of ball nodes to $cliques\_new$ ($act3$).

Listing 6: Event $Clique+$, in M1

EVENT $Clique+$
ANY B, x
WHERE
$grd1$: $B \subseteq cliques\_out$
$grd2$: $x \in B$
$grd3$: $B \subseteq (adjacents(x \mapsto t) \cup \{x\})$
$grd4$: $\forall y, z \cdot y \in B \land z \in B \land y \neq z \Rightarrow y \mapsto z \in g(t)$
$grd5$: $(\forall r \cdot r \in cliques\_out \land r \in adjacents(x \mapsto t) \land$
$\{r\} \times (B \setminus \{r\}) \subseteq g(t) \Rightarrow r \in B)$
$grd6$: $card(B) \geq 3$
$grd7$: $final = 0 \land t \neq tn$
THEN
$act1$: $cliques\_out := cliques\_out \setminus B$
$act2$: $cliques\_in := cliques\_in \cup B$
$act3$: $cliques\_new := cliques\_new \cup \{\{a \cdot a \in B | a\}\}$
END

• **Event *"Clique-"* :** The purpose of this event is to detect a node (noted *"x"*) which can not belong to any maximal clique. If this node belongs to a connected ball *B1*, *B1* should contain less than three nodes. Also, all the neighbours of *"x"*, which have not been affected to a maximal clique yet, have a ball including at most two nodes.

•**Event *"Oneshot"* :** This event refines the *Oneshot* presented in M0 to verify that the final value of $cliques\_new$ represents the result of the algorithm (see Listing 7). To do so, we reinforce the guard component by specifying that all maximal cliques in the graph *g* have been detected (grd2). By means of the theorem *Th4*, we verify that the set $cliques\_new$ represents the detected maximal and disjoint cliques. The abstract parameter *"c"*, defined in M0, is replaced with a concrete value ($cliques\_new$) by means of a witness. A witness designates a simple equality predicate involving the abstract parameters.

Listing 7: Event *Oneshot*, in M1

EVENT $Oneshot$
REFINES $Oneshot$
WHERE
$grd1 : solution(t) = \varnothing$
$grd2 : \forall y \cdot y \subseteq g(t) \wedge y \notin cliques\_new \implies y \notin cliques$
$grd3 : final = 0 \wedge t \neq tn$
$Th4 : cliques\_new \in combination$
$grd4 : \forall x \cdot x \in V \Rightarrow x \notin nodes\_alert$
WITH $c : c = cliques\_new$
THEN
$act1 : solution(t) := cliques\_new$
END

At this level, we refine the event $Remove\_Edge$ in two events:
• **Event *"Remove_Graph_Edge"* :** It specifies the case of deletion of a graph edge. To do so, we add a condition ($grd4$) to indicate that the edge $x \mapsto y$ does not belong to a maximal clique (noted n):
$\forall n \cdot n \in cliques\_new \Rightarrow \neg(x \in n \wedge y \in n)$
• **Event *"Remove_Clique_Edge"* :** This event, depicted in Listing 8, specifies the preliminary influence of the deletion of an edge belonging to a maximal clique. Formally, we reinforce the guard component to express that the edge $x \mapsto y$ belongs to a maximal clique noted "k" ($grd3$). Also, no topological event affects the clique ($grd4$). In the clause "THEN", we introduce two actions to update the sets of adjacent nodes of x and y ($act4$). In addition, we add x and y to the set $nodes\_alert$ ($act5$).

Listing 8: Event $Remove\_Clique\_Edge$, in M1

EVENT $Remove\_Clique\_Edge$
REFINES $Remove\_Edge$
ANY x, y, k
WHERE
...
$grd3 : k \in cliques\_new \wedge x \in k \wedge y \in k$
$grd4 : x \notin nodes\_alert \wedge y \notin nodes\_alert$
THEN
...
$act4 : adjacents := adjacents \Leftarrow \{(x \mapsto t) \mapsto adjacents(x \mapsto t) \setminus \{y\}, (y \mapsto t) \mapsto adjacents(y \mapsto t) \setminus \{x\}\}$
$act5 : nodes\_alert := nodes\_alert \cup \{x, y\}$
END

To specify the different situations of reinitialization of the neighbours (of the concerned maximal clique) of a node (noted u) belonging to the set $nodes\_alert$, we distinguish four cases. Because of space limitation, we only detail the informal description of these cases:

Case 1: It specifies the simplest case which consists in resetting the nodes states of u and its neighbours of the maximal clique. This case is specified by a new event called $Initialize1$.

Case 2: If there is a neighbour node of u (noted h) which can not belong to any maximal clique, we also reset the state of h. Indeed, the node h can be a member of a maximal clique after resetting the nodes states of the clique containing u. This situation is specified using a new event $Initialize2$.

Case 3: If a neighbour node of u (noted b) which belongs to another detected clique can form a maximal clique with the node b and other neighbours of the clique, then we introduce b to the set $nodes\_alert$. We specify this case by means of an event called $Initialize3$.

Case 4: The presence of the cases (2) and (3) requires the application of an event called $Initialize4$.

3) The third level : The refinement of M1 called M2 introduces more details about the algorithm CMCDN. In fact, we introduce the variable "$Clique$" as a function which assigns to each node the set of nodes of its maximal clique ($inv1 : Clique \in V \times (0..t) \rightarrow \mathbb{P}(V)$). Initially, the $Clique$ of each node is empty. To link the states between the machines M1 and M2, we define some gluing invariants:
◦ Each node of $cliques\_in$ belongs to a detected maximal clique.
(inv2) $\forall x, ti \cdot x \in cliques\_in \Rightarrow Clique(x \mapsto ti) \neq \varnothing \wedge x \in Clique(x \mapsto ti)$
◦ Each node of $cliques\_out$ has not computed its maximal clique yet.
(inv3) $\forall x, ti \cdot x \in cliques\_out \Rightarrow Clique(x \mapsto ti) = \varnothing$
◦ Each node which has not been assigned to a maximal clique yet belongs to the set $out\_cliques$.
(inv4) $\forall x, ti \cdot Clique(x \mapsto ti) = \varnothing \Rightarrow x \in cliques\_out$
◦ A node "x", which does not belong to a maximal clique, is not part of the set $nodes\_alert$.
(inv5) $\forall x, ti \cdot Clique(x \mapsto ti) = \varnothing \vee Clique(x \mapsto ti) = \{x\} \Rightarrow x \notin nodes\_alert$

At this refinement level, the events of the previous level still exist but they become more concrete. We restrict to detail in what follows some events :
• **Event *"Remove_Graph_Edge"* :** We refine the event $Remove\_Graph\_Edge$ by reinforcing the guard component. In fact, we replace the guard $grd4$ using the variable $Clique$ that the removed edge does not belong to any maximal clique: $grd4 : \neg(Clique(x \mapsto t) = Clique(y \mapsto t) \wedge card(Clique(x \mapsto t)) \geq 3)$
• **Event *"Clique+"*:** The goal of this event is to detect maximal and disjoint cliques and assign to each node its corresponding clique (see Listing 9).

Listing 9: Event $Clique+$, in M2

EVENT $Clique+$
REFINES $Clique+$
ANY B, x
WHERE
$grd1 : Clique(x \mapsto t) = \varnothing \wedge B \cap \{k \cdot Clique(k \mapsto t) \neq \varnothing | k\} = \varnothing$
...
$grd5 : \forall r \cdot Clique(r \mapsto t) = \varnothing \wedge r \in adjacents(x \mapsto t) \wedge \{r\} \times (B \setminus \{r\}) \subseteq g(t) \Rightarrow r \in B$
$grd6 : card(inter(\{a \cdot a \in B | adjacents(a \mapsto t) \cup \{a\}\}) \setminus \{k \cdot Clique(k \mapsto t) \neq \varnothing | k\}) \geq 3 \wedge finite(inter(\{a \cdot a \in B | adjacents(a \mapsto t) \cup \{a\}\}) \setminus \{k \cdot Clique(k \mapsto t) \neq \varnothing | k\})$
$grd7 : final = 0 \wedge t \neq tn$
THEN
$act1 : Clique := Clique \Leftarrow \{a \cdot a \in B | (a \mapsto t) \mapsto B\}$
END

In fact, we reinforce the guard component by using the new variable $Clique$. The guard ($grd1$) specifies that the center x

of the ball *B* and its neighbours from *B* do not belong to any detected clique. The *grd5* states that *B* can not be extended by other nodes which have not been assigned to maximal cliques yet. We indicate in the guard $grd6$ that the intersection of all the elements of *N(a)*, which do not belong to maximal cliques, contains at least three nodes. We note *"a"* as each node of the ball *B*. In the action component, we set the maximal clique of each node of the ball to *"B"* ($act1$).

• **Event *"Oneshot"*:** This event refines the *"Oneshot"* presented in the machine M1. It uses the concrete variable $Clique$ to check that each node of the graph has computed the maximal clique to which it belongs. In fact, the result of the algorithm represents the set containing the maximal clique of each node x (if $Clique(x \mapsto t) \neq \{x\}$).

4) The fourth level : Once the machine of the third level has been specified and proven, it can be refined for describing the local label modification and encoding the relabeling rules proposed in Section II. In order to reach this goal, we introduce a new variable "$State$" ($inv1 : State \in V \times (0\,..\,t) \rightarrow labels$) which assigns to each node a label from the set *"labels"* that encodes the state of a process. Initially, all the nodes are labeled *"Init"*. The addition of the variable "$State$" involves adding new properties which link the abstract state variables to the concrete ones. We have formalized these properties in the form of Event-B invariants:

◦ A node x, which has $Clique(x \mapsto ti)$ not empty, belongs to a maximal clique or it is an isolated node at the date ti.

(inv2) $\forall x, ti \cdot Clique(x \mapsto ti) \neq \varnothing \Rightarrow$
$x \in State^{-1}[\{C, I, A\}]$

◦ A node which has not been assigned to a maximal clique yet is in the initial or the waiting state.

(inv3) $\forall x, ti \cdot Clique(x \mapsto ti) = \varnothing \Rightarrow$
$x \in State^{-1}[\{Init, W\}]$

◦ If a node is labeled C, its maximal clique contains itself and a set of its neighbours.

(inv4) $\forall x, ti \cdot State(x \mapsto ti) = C \Rightarrow Clique(x \mapsto ti) \subseteq \{x\} \cup adjacents(x \mapsto ti) \wedge x \in Clique(x \mapsto ti)$

◦ Each maximal clique contains one center node labeled C and the other nodes are labeled I.

(inv5) $\forall x, y, ti \cdot y \in Clique(x \mapsto ti) \setminus \{x\} \wedge State(x \mapsto ti) = C \Rightarrow State(y \mapsto ti) = I \wedge Clique(y \mapsto ti) = Clique(x \mapsto ti)$

◦ If a node y is labeled *I*, it has a neighbouring node which belongs to the same maximal clique and it is the center of this clique.

(inv6) $\forall y, ti \cdot State(y \mapsto ti) = I \Rightarrow (\exists x \cdot x \in adjacents(y \mapsto ti) \wedge State(x \mapsto ti) = C \wedge Clique(y \mapsto ti) = Clique(x \mapsto ti))$

◦ An isolated node does not belong to any maximal clique, then its maximal clique is the identity.

(inv7) $\forall x, ti \cdot State(x \mapsto ti) = A \Rightarrow Clique(x \mapsto ti) = \{x\}$

◦ Each node x labeled C or I belongs to a maximal clique, then the set of elements of its clique is not empty.

(inv8) $\forall x, ti \cdot State(x \mapsto ti) \in \{C, I\} \Rightarrow Clique(x \mapsto ti) \neq \varnothing$

◦ Each node in the initial or the waiting state has not been assigned to a maximal clique yet.

(inv9) $\forall x, ti \cdot State(x \mapsto ti) \in \{W, Init\} \Rightarrow Clique(x \mapsto ti) = \varnothing$

At this level, we refine the event *"Oneshot"* and we specify the six relabeling rules of our algorithm:

◦ The event "$Rule1$" refines the event "$Clique+$" defined in M2 to specify the rule *R1*.

◦ We introduce a new event called *"Rule2"* to specify the rule *R2*.

◦ The event "$Rule3$" specifies the rule *R3* and refines the event $Clique-$ of the machine M2.

◦ The event "$Rule4$" refines the event $Remove\_Graph\_Edge$ to express the modification made when deleting an edge of the graph.

◦ The event "$Rule5$" refines the event $Remove\_Clique\_Edge$ to specify the preliminary influence of the removal of an edge belonging to a maximal clique.

◦ The events ($Rule6$,$Rule6'$,$Rule6''$,$Rule6'''$) refine respectively the events ($Initialize1$,..., $Initialize4$) to express the different cases of the rule $R6$.

◦ The event *"Oneshot"* verifies that, at the end of the algorithm execution, no node is in the initial (labeled *"Init"*) or the waiting state (labeled *"W"*).

C. Proof statistics

To prove the correctness of our formal model, a number of proof obligations (POs) generated by the Rodin platform should be discharged. The algorithm development results in 405 POs, in which 226 (56%) POs are proved automatically and 179 (44%) are proved interactively using the RODIN prover. An Event-B model is correct when all POs have been discharged. Formal definitions of all POs are given in [1].

IV. Conclusion

We have presented in this paper a new distributed algorithm for enumerating maximal cliques in dynamic networks. Our algorithm combines local computations model and refinement to prove its correctness. The proposed algorithm is based on six relabeling rules which allow to detect maximal cliques and react correctly in case of edge deletion.

References

[1] J.-R. Abrial, *Modeling in Event-B - System and Software Engineering*. Cambridge University Press, 2010.

[2] A. Conte, R. D. Virgilio, A. Maccioni, M. Patrignani, and R. Torlone, "Finding all maximal cliques in very large social networks," in *the International conference Extending Database Technology (EDBT)*, 2016, pp. 173–184.

[3] A. Ferreira, "On models and algorithms for dynamic communication networks: The case for evolving graphs," in *the conference ALGOTEL*, 2002.

[4] G. T. Leavens, J.-R. Abrial, D. Batory, M. Butler, A. Coglio, K. Fisler, E. Hehner, C. Jones, D. Miller, S. Peyton-Jones, M. Sitaraman, D. R. Smith, and A. Stump, "Roadmap for enhanced languages and methods to aid verification," in *the International Conference Generative Programming and Component Engineering (GPCE)*. ACM, 2006, pp. 221–236.

[5] I. Litovsky, Y. Métivier, and E. Sopena, "Handbook of graph grammars and computing by graph transformation." World Scientific, 1999, ch. Graph Relabelling Systems and Distributed Algorithms, pp. 1–56.

[6] C. Luo, J. Yu, D. Yu, and X. Cheng, "Distributed algorithms for maximum clique in wireless networks," in *the International conference Mobile Ad-hoc and Sensor Networks (MSN)*. IEEE, 2015, pp. 222–226.

[7] N. A. Lynch, *Distributed algorithms*. Elsevier, 1996.

[8] Y. Xu, J. Cheng, and A. W.-C. Fu, "Distributed maximal clique computation and management," *IEEE Transactions on Services Computing*, vol. 9, no. 1, pp. 110–122, 2016.

.