Strengthening Topological Conditions for Relabeling Algorithms in Evolving Graphs

Florent Marchand de Kerchove, Frédéric Guinand

Strengthening Topological Conditions for Relabeling Algorithms in Evolving Graphs

Florent Marchand de Kerchove – fmdkdd@gmail.com
Normandy Universtiy ULH LITIS, France
and
Frédéric Guinand – frederic.guinand@univ-lehavre.fr
Normandy University ULH LITIS, France

We use the framework of relabeling algorithms over evolving graphs by Casteigts, Chaumette and Ferreira in 2012 to ground our contributions to the domain of distributed algorithms, which are twofold: (1) we provide a sufficient condition for the decentralized counting algorithm proposed by Casteigts in its PhD dissertation, (2) we define a tightness criterion for sufficient and necessary conditions.

Categories and Subject Descriptors:

Additional Key Words and Phrases: Relabeling algorithms, evolving graphs, counting

1. CONTEXT

1.1 Dynamic graphs

Graphs are powerful mathematical structures, suited to numerous theoretical and real-world applications. However, for all their usefulness in a static context, graphs are not sufficient when it comes to modeling dynamic relationships between objects. Dynamic graphs arose from a need to include time into graph-based models.

Straightforwardly, a dynamic graph can be decomposed as a discrete sequence of static graphs. Each static graph is a snapshot of the dynamic graph at a given time. These static graphs can then be studied by all the mathematical tools already at our disposal. This is precisely the view adopted by Ferreira and his "evolving graph" model [2004].

Born in the context of mobile ad-hoc networks, the formalism of evolving graphs is a mathematical tool to analyze a sequence of static graphs captured from the same dynamic network. It introduces new concepts required by the incorporation of time, like the one of a path over time, called a journey. Our contribution is deeply rooted in this formalism, and as a matter of fact we will make use of the definitions presented in this seminal article.

1.2 Related work

The interest in the study of distributed algorithms resulted in a large number of models to characterize distributed algorithms, and the communication networks they are usually executed on. These models are seldom compatible in their assumptions on *synchronizations protocols* and *mobility models* for example. As a consequence, algorithms that are elaborated for distinct, specific models are not easily comparable to each other. This diversity called for higher-order theoretical frameworks, where the intrinsic properties of distributed algorithms can be analyzed without getting dragged into implementation details.

We focus on the model introduced by Casteigts [2007], where graph relabeling systems [1999] are coupled with evolving graphs [2004]. Although superficially it may appear similar to the rather popular approach of population protocols introduced by Angluin et al. [2006], the assumptions and implications are radically different. For one, here the interactions between nodes are finite; no assumptions on the recurrence of these interactions are made. That is why the analytical approach required to study algorithms under this model is necessarily different to that of population protocols. Thus, even though algorithms expressed here may have already appeared in the context of population protocols, their analysis must not be confused.

This framework was later extended by Casteigts et al. [2009; 2012]. While these works are fairly recent, they have already been applied to the spanning tree maintenance problem by Pigné et al. [2010] and to the mutual exclusion problem by Floriano et al. [2011].

The next section is devoted to the definitions and results from Casteigts et al. [2012] required by our contributions, which are presented in the third section.

2. PRELIMINARIES

Our contributions build upon the work established by Casteigts et al. [2012]. Therefore, the concepts of *local computation*, *graph relabelings* and *evolving graphs* are required to fully understand the results of next section.

For the sake of clarity and completeness, we reproduce here the definitions by Casteigts et al. [2012] needed to express our results.

2.1 Graph relabelings

Graph relabelings [1999] are a formalism where distributed algorithms are represented as a set of local interaction rules. These rules are independent from any communication protocol. Abstracting the effective communication allows us to specify and reason about important properties of distributed algorithms, such as correctness and termination, without limiting these results to a specific implementation.

As the name implies, a graph relabeling system is first and foremost a graph with labels on its vertices and edges. These labels are used by the interaction rules of the *relabeling algorithm*. An interaction rule is defined as a transition from one pattern of vertices and edges and their associated labels (*preconditions*), to another such pattern (*actions*). Since graph relabeling systems were introduced to characterize properties of local computation, interaction rules are local: they involve a limited set of connected vertices and edges. In this paper, we only consider

interaction rules between pairs of connected vertices, i.e. neighbors.

We now give a formal definition of graph relabeling systems. Let $G = (V_G, E_G)$ be a finite undirected loopless graph, with V_G as the set of nodes in our network, and E_G representing the set of communication links between them. Two vertices u and v are *neighbors* if and only if they share a common edge (u, v) in E_G.

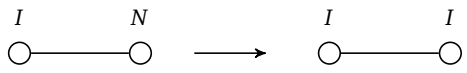
Let $\lambda : V_G \cup E_G \to L^*$ be a mapping that associates every vertex and edge from G with one or several labels from an alphabet L. The label of a given vertex or edge u, at a given time t is denoted by $\lambda_t(u)$. The pair (G, λ) is the *labeled graph*, written $\boldsymbol{G}$.

A complete algorithm is defined by the triplet $\{L, I, P\}$, where I is the set of initial states, and P is a set of *relabeling rules*. Since we are interested in distributed algorithms, where every node in the network will execute the same identical algorithm, the set of rules P is the same for all vertices.

Algorithm 1 (or A_1 for short) is an example of a complete algorithm expressed in the formalism of graph relabelings. Each vertex can be in one of two states: I and N, standing for *informed* and *non-informed* respectively. Initially, only the emitter vertex has the I state. Then, repeated application of the rule diffuses the information in the network. This is a simple and general information propagation algorithm.

Algorithm 1: Information propagation

Initial states: I for the emitter, N for every other node

r_1: 

The algorithm works in the following way. Since rules are patterns, each node looks for a match in its neighborhood in order to follow the rule's preconditions (the left-hand side of the rule). When a pair matches, both nodes modify their labels in the way described by the right-hand side of the rule. Figure 1 gives a step-by-step example of executing algorithm 1 on a static graph.

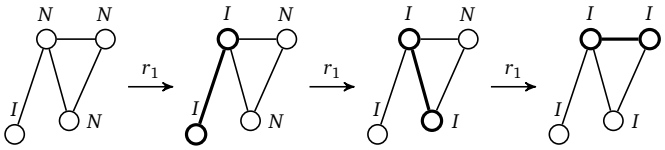


Fig. 1. Example execution of the information propagation relabeling algorithm on a static graph. Here, all nodes are informed at the end. Note that this is only one possible execution sequence of the algorithm for this graph; others appear when matching pairs of nodes are selected in a different order.

Note that, from a node's perspective, when two or more neighbor nodes match a rule's preconditions, only one rule can be applied at a time. Depending on the order the rules are applied to these matching pairs, the algorithm may have different outcomes.

2.2 Evolving graphs

2.2.1 *Definition.* Evolving graphs were introduced by Ferreira [2004] as a model for dynamic networks. In this model, the evolution of the network topology is simply recorded as a sequence of static graphs, where each static graph can be seen as a snapshot of the network at a given time.

Formally, an evolving graph is a triplet $(G, S_G, S_\mathbb{T}) = \mathcal{G}$, where:

—$S_\mathbb{T}$ is the sequence of dates used to capture the static graphs. $\mathbb{T}$ can be anything meaningful to the network: discrete ($\mathbb{T} \subset \mathbb{N}$) and continuous ($\mathbb{T} \subset \mathbb{R}$) time systems are common.

—S_G is the sequence of undirected static graphs $G_i = (V_i, E_i)$, where G_i is a snapshot of the network topology during an interval $[t_i, t_{i+1})$.

—G is the union of all G_i in S_G, called the *underlying graph* of $\mathcal{G}$.

We will use the simple notations V and E to denote the sets of vertices and edges of the underlying graph G. A vertex (resp. edge) u is in V (resp. E) if and only if it belongs to at least one static graph in S_G. In addition, we will use the notation $\mathcal{G}_{[t_a,t_b)}$ when taking a *temporal subgraph* $\mathcal{G}' = (G', S'_G, S'_\mathbb{T})$ of $\mathcal{G} = (G, S_G, S_\mathbb{T})$, where $G' \subseteq G$, $S'_G = \{G_i \in S_G : t_i \in [t_a, t_b)\}$, and $S'_\mathbb{T} = \{t_i \in S_\mathbb{T} \cap [t_a, t_b)\}$.

Note that evolving graphs are a *post-mortem* view of the network; in a given evolving graph, all the nodes that can appear or disappear belong to V. As such, even if a node can be seen as "appearing" from another node's viewpoint, for us it was present all along.

2.2.2 *Basic concepts.* First, we consider a *presence function* $\rho : E \times \mathbb{T} \to \{0, 1\}$ that indicates whether a given edge is present at a given date. For $e \in E$ and $t \in [t_i, t_{i+1})$, with t_i and t_{i+1} being two consecutive dates in $S_\mathbb{T}$, $\rho(e, t) = 1 \iff e \in E_i$.

A *journey* is a path *over time* between two vertices. Formally, a journey in $\mathcal{G}$ is a sequence of couples $J = \{(e_1, \sigma_1), (e_2, \sigma_2), \ldots, (e_k, \sigma_k)\}$ where $\{e_1, e_2, \ldots, e_k\}$ is a walk in G, $\{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ is a non-decreasing sequence of dates from $\mathbb{T}$, and $\rho(e_i, \sigma_i) = 1$ for all $i \leq k$. A *strict journey* only contains couples (e_i, σ_i) taken from distinct graphs of the sequence S_G.

For any u, v in V, if a journey from u to v exists in $\mathcal{G}$, we write $u \rightsquigarrow v$, or $u \overset{\text{st}}{\rightsquigarrow} v$ in the case of a strict journey. We assume that $u \rightsquigarrow u$ for all u in V. Note that a journey is not necessarily symmetrical, even if edges are undirected, because time intervals create a new level of direction. The *horizon* of a node u is the set $\{v \in V : u \rightsquigarrow v\}$, thus u is included in its own horizon.

2.3 Relabelings over evolving graphs

Now we combine graph relabelings and evolving graphs to create an analysis framework for distributed algorithms on dynamic networks.

Let $\mathcal{G} = (G, S_G, S_\mathbb{T})$ be an evolving graph. The static graph in S_G that covers the time interval $[t_i, t_{i+1})$ is written G_i; we have $G_i \in S_G$ and $t_i, t_{i+1} \in S_\mathbb{T}$. The labeled graph $(G_i, \lambda_{t_i+\epsilon})$, denoted $\boldsymbol{G}_i$, represents the state of the network *just after* the topological event of date t_i, and $\boldsymbol{G}_{i[}$ denotes the labeled graph $(G_{i-1}, \lambda_{t_i-\epsilon})$ representing the network state *just before* that event. Thus, $\boldsymbol{G}_i = \text{Event}_{t_i}(\boldsymbol{G}_{i[})$, where Event_{t_i} is the topological event occuring at time t_i and mapping the static graph $G_{t_{i-1}}$ to G_{t_i}.

Between two consecutive topological events, any number of relabelings may take place. For a given algorithm A and two consecutive dates $t_i, t_{i+1} \in S_\mathbb{T}$, we denote by $R_{A_{[t_i,t_{i+1})}}$ one of

the possible relabeling sequence induced by A on the graph G_i during the period $[t_i, t_{i+1})$. We have $\boldsymbol{G}_{i+1[} = R_{A_{[t_i,t_{i+1})}}(\boldsymbol{G}_i)$.

Let us call t_k the date of the last static graph in S_G. A complete execution sequence from t_0 to t_k is then given by an alternated sequence of relabeling steps and topological events:

$$X = R_{A_{[t_{k-1},t_k)}} \circ \text{Event}_{t_{k-1}} \circ \cdots \circ \text{Event}_{t_1} \circ R_{A_{[t_0,t_1)}}(\boldsymbol{G}_0)$$

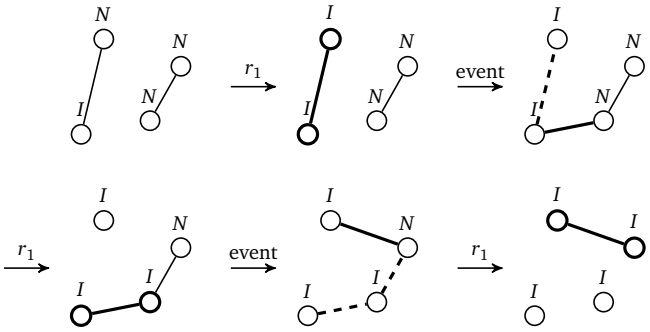


Fig. 2. Execution of the information propagation relabeling algorithm on an evolving graph. This time, the execution is a sequence of intertwined relabeling steps and topological events. In future diagrams, we will combine relabeling steps and topological events for the sake of brevity; thus we will only show the sequence of labeled graphs.

We already mentioned that the order of execution of the rules is not deterministic, since it depends on implementation details concerning the selection of pairs of nodes matching a rule's preconditions. We denote by $\chi_{A/\mathcal{G}}$ the set of all possible execution sequences of an algorithm A over an evolving graph $\mathcal{G}$. An example execution of a relabeling algorithm over an evolving graph is given in figure 2.

2.4 Analysis of distributed algorithms

A distributed algorithm can have multiple outcomes for the same graph. We usually want the algorithm to be complete: it should achieve its goal in all cases. For example, the propagation algorithm (algorithm 1) should inform all nodes in the network. Unfortunately, all networks will not necessarily allow the algorithm to complete. If the network contains at least two connected components, then only one of them will have all of its nodes informed, since there is only one informed node initially, and information is propagated along edges.

Hence, we find a first way of analyzing distributed algorithms: by characterizing graphs on which they are complete, and graphs on which they can never reach their goal. A condition on graphs ensuring the completeness of algorithm A will be called a sufficient condition for A. Conversely, a necessary condition for A defines the class of graphs on which A will always fail.

Formal definitions of these concepts follow.

2.4.1 *Objectives of an algorithm.* Given an algorithm A and a labeled graph $\boldsymbol{G}$, the state one wishes to reach can be given by a logic formula P on the labels of vertices (and edges, if appropriate). In the case of the propagation algorithm, such a terminal state could be that all nodes are informed,

$$P_1(\boldsymbol{G}) = \forall v \in V, \lambda(v) = I$$

The objective O_A is then defined as the fact of verifying the desired property by the end of the execution, that is, on the final labeled graph, after the last relabeling step.

$$O_{A_1} = P_1(\boldsymbol{G}_{k[})$$

2.4.2 *Necessary conditions.* Given an algorithm A, its objective O_A and an evolving graph property C_N, the property C_N is a *(topology-related) necessary condition* for O_A if and only if:

$$\forall \mathcal{G}, \neg C_N(\mathcal{G}) \implies \neg O_A$$

Proving this result comes to prove that

$$\forall \mathcal{G}, \neg C_N(\mathcal{G}) \implies \nexists X \in \chi_{A/\mathcal{G}} | P(\boldsymbol{G}_{k[})$$

In other words, the desired state is not reachable by the end of the execution (time k), unless the condition is verified.

2.4.3 *Sufficient conditions.* Symmetrically, an evolving graph property C_S is a *(topology-related) sufficient condition* for A if and only if:

$$\forall \mathcal{G}, C_S(\mathcal{G}) \implies O_A$$

Proving this result comes to prove that

$$\forall \mathcal{G}, C_S(\mathcal{G}) \implies \forall X \in \chi_{A/\mathcal{G}} | P(\boldsymbol{G}_{k[})$$

The desired state is always reached by the end of the execution if the condition holds.

Because we have not made any assumptions on the synchronization between nodes, we have no way of ensuring that a rule will effectively be applied. Therefore, Casteigts et al. [2012] formulate a progression hypothesis that enables the characterization of sufficient conditions.

PROGRESSION HYPOTHESIS 1. *In every time interval $[t_i, t_{i+1})$, with t_i in $S_{\mathbb{T}}$, each vertex is able to apply at least one relabeling rule with each of its neighbors, provided the rule preconditions are* already *satisfied at time t_i (and still satisfied at the time the rule is applied).*

2.4.4 *Analysis of the propagation algorithm.* Casteigts et al. show [2012] that the information propagation algorithm has the following necessary and sufficient conditions.

CONDITION 1. $\exists u \in V : \forall v \in V, u \rightsquigarrow v$.
(There is a node that can reach all the others by a journey.)

CONDITION 2. $\exists u \in V : \forall v \in V, u \overset{st}{\rightsquigarrow} v$.
(There is a node that can reach all the others by a strict journey.)

3. CONTRIBUTIONS

In this section, we present our core contributions to the analysis of distributed algorithms on dynamic graphs.

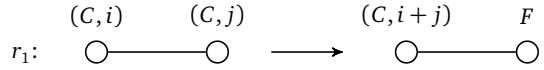
First, we provide a sufficient condition for the decentralized counting algorithm from Casteigts [2007]. A necessary condition for this algorithm was formulated in Casteigts et al. [2012], but a sufficient condition was left open. We then introduce the concept of *tight* conditions, to strengthen the guarantees offered by necessary and sufficient conditions. Finally, we review the conditions and algorithms introduced so far with respect to this tightness criterion.

3.1 Sufficient condition for the decentralized counting algorithm

3.1.1 *Decentralized counting algorithm.* This decentralized counting algorithm was first proposed in Casteigts' thesis [2007], along with other variants.

Algorithm 2: Decentralized counting

Initial states: $(C,1)$ for every node



This is an example of a truly distributed algorithm: all nodes have the same initial state, and all nodes execute the same algorithm. Each node has two labels: a state indicator (C or F), and a counter. Initially, a node starts in the "counting" state (C), with a counter of value 1. The counter indicates the number of counted nodes so far by its holder, hence they all begin at 1. When two nodes in the counting state meet and apply rule 1, one of them transitions to the "counted" state (F), and the other updates the value of its counter by summing the counters of both nodes. Ultimately, the goal is to have the last node remaining in the counting state to hold the total number of nodes in the network as the value of its counter. Figure 3 gives an example execution sequence.

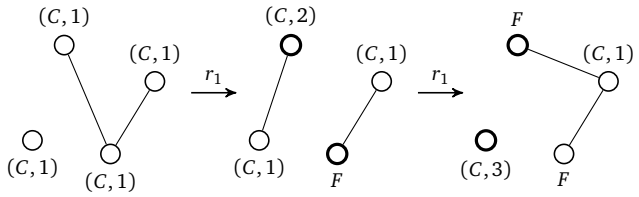


Fig. 3. Example execution of the decentralized counting algorithm. If we consider this to be the whole execution sequence, then the counting was not successful: two nodes remain in the counting state. For these two nodes to apply rule 1 and merge, there must be an edge between them; this is the intuition behind a sufficient condition for this algorithm.

Casteigts showed [2007] that this algorithm has the following invariant. Let $\mathscr{C}$ (resp. $\mathscr{F}$) be the set of nodes in state "C" (resp. "F"), and V be the set of all nodes. Then $|\mathscr{C}|+|\mathscr{F}| = |V|$ holds at any time of the computation. It follows that if $\mathscr{C} = \{u\}$ (only one counting node remains), then u's counter value is equal to $|V|$.

We can express the objective of this algorithm as the property that only one counting node remains at the end of the computation. Formally:

$$P_2 = \exists u \in V : \forall v \in V \setminus \{u\}, s(u) = C \wedge s(v) = F$$
$$O_{A_2} = P_2(\boldsymbol{G}_{k[})$$

Where $s(u)$ is the "state" (here in $\{C,F\}$) of the node u. The necessary condition to this algorithm was then shown to be:

Condition 3. $\exists v \in V : \forall u \in V, u \rightsquigarrow v$.
(There is a node reachable by all the others.)

3.1.2 *Sufficient condition.* Let us now prove that condition 4 is a sufficient condition for algorithm 2.

Condition 4. *The underlying graph G is complete. Precisely:* $\forall u, v \in V,\ u \neq v \Longrightarrow (u,v) \in E$.

First, note the following properties of algorithm 2.

Lemma 1. $\forall t_i, t_j \in S_{\mathbb{T}},\ j \leq i,\ \forall u \in V,\ s_{t_i}(u) = C \Longrightarrow s_{t_j}(u) = C$.
("C" labels never change until they disappear.)

Proof. Counters can only disappear from a vertex by application of r_1. Any counter still present at $t_i \in S_{\mathbb{T}} \setminus \{t_0\}$ must have been there from the beginning. □

Lemma 2. $\forall t_i, t_j \in S_{\mathbb{T}},\ j \geq i,\ \forall u \in V,\ s_{t_i}(u) = F \Longrightarrow s_{t_j}(u) = F$.
("F" labels never change once they appear.)

Proof. No rule can apply to a vertex with a "F" label, thus its label can never change once it becomes "F". □

Lemma 3. *Under progression hypothesis 1,* $\forall t_i \in S_{\mathbb{T}} \setminus \{t_k\}$, $\forall (u,v) \in E_i,\ s_{t_i}(u) = s_{t_i}(v) = C \Longrightarrow F \in s_{t_{i+1}[}(\{u,v\})$.
(If two counters share an edge, at least one of them will disappear.)

Proof. During the relabeling step $R_{[t_i,t_{i+1})}$, either r_1 is applied to (u,v), or r_1 can not be applied because preconditions on the labels are not met anymore after an intermediary relabeling. In the first case, one counter disappears from one vertex of $\{u,v\}$; in the second case, one vertex of $\{u,v\}$ already lost its counter. In both cases, $F \in s_{t_{i+1}[}(\{u,v\})$. □

We can now show that condition 4 (C_4) is sufficient for algorithm 2 to fulfill its objective.

Proposition 1. *Condition 4 is sufficient on G to guarantee that algorithm 2 will reach its objective O_{A_2}.*

Proof. (By contradiction). Assume O_{A_2} is not satisfied: for some execution sequence $X \in \chi_{A_2/\mathcal{G}}$, there are at least two final counters in $\boldsymbol{G}_k$. Let u and v be two nodes with such counters: $u,v \in V,\ u \neq v,\ s_{t_k}(u) = s_{t_k}(v) = C$. Since G is complete (by condition 4), $(u,v) \in E_i$, for some $t_i \in S_{\mathbb{T}}$. By lemma 1, $s_{t_i}(u) = s_{t_i}(v) = C$. Then, by lemma 3, either $s_{t_{i+1}[}(u) = F$ or $s_{t_{i+1}[}(v) = F$, and in both cases we have $F \in s_{t_k}(\{u,v\})$ by lemma 2, leading to a contradiction. Hence, $C_4 \Longrightarrow O_{A_2}$. □

3.2 Tight necessary and sufficient conditions

A necessary condition might be broader than required. We know that any graph not satisfying this condition will never fulfill the algorithm's objective, but it gives us no additional clue about graphs *satisfying* the condition. Indeed, for any graph satisfying the condition, some may never fulfill the objective, some may fulfill it in some cases, and some may fulfill it in all cases; we can not know without a stronger definition.

Besides, it is not difficult to find trivial necessary conditions that are overly broad. For any relabeling algorithm, any condition which encompass all graphs is a necessary condition. Take the condition that the graph should have nodes: $|V| > 0$. Then, let S be the set of graphs not satisfying this condition; trivially, $S = \emptyset$. It follows that any graph in S will never fulfill the objective, since S is empty, hence $(|V| > 0)$ is a necessary condition.

It is trivial in the sense that it does not enlarge the set of graphs for which the algorithm will fail.

When finding necessary conditions, we preferably want them to be as tight as possible. A necessary condition should partition the set of dynamic graphs into two subsets: the set of all graphs for which the algorithm will systematically fail, and the set of all graphs for which the algorithm will succeed at least once. To this end, we define *tight necessary conditions* that generates such partitioning.

DEFINITION 1. *Let A be an algorithm, O_A be its objective, and C_N be a necessary condition. C_N is a tight necessary condition if and only if*

$$\forall \mathcal{G}, C_N(\mathcal{G}) \Longrightarrow \exists X \in \chi_{A/\mathcal{G}} | P(\boldsymbol{G}_k)$$

In other words, if C_N holds for an evolving graph $\mathcal{G}$, then at least one execution sequence of A over $\mathcal{G}$ will reach the desired state.

Symmetrically, we define tight sufficient conditions.

DEFINITION 2. *Let A be an algorithm, O_A be its objective, and C_S be a sufficient condition. C_S is a tight sufficient condition if and only if*

$$\forall \mathcal{G}, \neg C_S(\mathcal{G}) \Longrightarrow \exists X \in \chi_{A/\mathcal{G}} | \neg P(\boldsymbol{G}_k)$$

In other words, if C_S does not hold for a graph $\mathcal{G}$, then at least one execution sequence of A over $\mathcal{G}$ will fail to reach the desired state.

Note that a condition that is both necessary and sufficient is also a tight necessary and tight sufficient condition.

3.2.1 *Tightening known conditions.* We now show that condition 4 is a tight sufficient condition for algorithm 2. We have two ways to do so:

—Show that an evolving graph lacking completeness of its underlying graph will fail to fulfill its objective in at least *one* outcome.

—Show that all graphs for which all outcomes succeed in fulfilling the objective have a complete underlying graph.

The following proof uses the former path.

PROOF. Let $\mathcal{G} = (G, S_G, S_{\mathbb{T}})$ be an evolving graph. By hypothesis, G is not complete; i.e. there are two distinct nodes $u, v \in V$ such that $(u, v) \notin E$. Since u and v are never neighbors, they can not apply rule r_1. If u and v are the only two counting nodes left, then it follows that $\neg P(\boldsymbol{G}_k)$, thus the objective O_{A_2} can not be fulfilled and condition 4 is tight. We are left with exposing a relabeling sequence which leaves u and v with two C labels.

When applying r_1, any of the two nodes can keep the C label, creating two possible outcomes. Every time r_1 is applied to u (resp. v) with another node, we choose the outcome where u (resp. v) keeps the C label. Ultimately, there are at least two nodes in the counting state at time t_k: u and v. There may be more, but in all cases the algorithm fails to fulfill the objective. □

It can also be shown that all necessary and sufficient conditions given for the algorithms in Casteigts et al. [2012] are tight. This is expected, because mathematical proofs have a tendency to follow Occam's razor; the smallest set of hypotheses needed by the proof is kept. Nonetheless, tight conditions ensure we narrowed down the right property required by the algorithm to fail (or succeed).

Conclusion

Throughout this article, we built upon the foundations provided by Casteigts et al. [2012] in the domain of distributed algorithms over dynamic graphs.

We showed that a complete underlying graph was sufficient for the decentralized counting algorithm to succeed. Then, we defined a tightness criterion to ensure the generality of necessary and sufficient conditions, and we proceeded to demonstrate the tightness of the sufficient condition we gave for the decentralized counting algorithm.

By partitioning the set of evolving graphs, this tightness criterion allows us to characterize maximal conditions: those that can not be improved. In turn, since tight conditions give the classes of graphs for which the algorithm will succeed (for a sufficient condition), or fail (for a necessary condition), they can be used to characterize the difficulty of an algorithm: a smaller class of graphs on which success is guaranteed indicates a more difficult algorithm.

Furthermore, we believe this concept of tight conditions can be applied to other analytical models of distributed computing, where its benefits would be of a great value.

REFERENCES

ANGLUIN, D., ASPNES, J., DIAMADI, Z., FISCHER, M., AND PERALTA, R. 2006. Computation in networks of passively mobile finite-state sensors. *Distributed Computing 18*, 235–253.

CASTEIGTS, A. 2007. Contribution à l'algorithmique distribuée dans les réseaux mobiles ad hoc - Calculs locaux et réétiquetages de graphes dynamiques. Ph.D. thesis.

CASTEIGTS, A., CHAUMETTE, S., AND FERREIRA, A. 2009. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *Proc. of 16th Intl. Conference on Structural Information and Communication Complexity*. Lecture Notes in Computer Science, vol. 5869. Springer-Verlag, 126–140.

CASTEIGTS, A., CHAUMETTE, S., AND FERREIRA, A. 2012. Distributed computing in dynamic networks: Towards a framework for automated analysis of algorithms. *CoRR*.

FERREIRA, A. 2004. Building a reference combinatorial model for MANETs. *Network, IEEE 18,* 5, 24–29.

FLORIANO, P., GOLDMAN, A., AND ARANTES, L. 2011. Formalization of the necessary and sufficient connectivity conditions to the distributed mutual exclusion problem in dynamic networks. In *10th IEEE International Symposium on Network Computing and Applications*. 203–210.

LITOVSKY, I., MÉTIVIER, Y., AND SOPENA, É. 1999. *Graph relabelling systems and distributed algorithms*. Vol. 3. Chapter 1, 1–56.

PIGNÉ, Y., CASTEIGTS, A., GUINAND, F., AND CHAUMETTE, S. 2010. Construction et maintien d'une forêt couvrante dans un réseau dynamique. In *12e Rencontres francophones sur les aspects algorithmiques de télécommunications*.