Robustness: A new form of heredity motivated by dynamic networks

Arnaud Casteigts, Swan Dubois, Franck Petit, John Robson

Version of Record: https://www.sciencedirect.com/science/article/pii/S0304397519304979
Manuscript_8b5ca57bda724c8be225378c2c98ff2b

Robustness: a New Form of Heredity Motivated by Dynamic Networks

Arnaud Casteigts

Université de Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, France

Swan Dubois

Sorbonne Université, CNRS, Inria, LIP6 UMR 7606, France

Franck Petit

Sorbonne Université, CNRS, Inria, LIP6 UMR 7606, France

John M. Robson

Université de Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, France

Abstract

We investigate a special case of hereditary property in graphs, referred to as *robustness*. A property (or structure) is called robust in a graph G if it is inherited by all the connected spanning subgraphs of G. We motivate this definition using two different settings of dynamic networks. The first corresponds to networks of low dynamicity, where some links may be permanently removed so long as the network remains connected. The second corresponds to highly-dynamic networks, where communication links appear and disappear arbitrarily often, subject only to the requirement that the entities are temporally connected in a recurrent fashion (*i.e.* they can always reach each other through temporal paths). Each context induces a different interpretation of the notion of robustness.

We start by motivating the definition and discussing the two interpretations, after what we consider the notion independently from its interpretation, taking as our focus the robustness of *maximal independent sets* (MIS). A graph may or may not admit a robust MIS. We characterize the set of graphs $\mathcal{RMIS}^{\forall}$ in which *all* MISs are robust. Then, we turn our attention to the graphs that *admit* a robust MIS ($\mathcal{RMIS}^{\exists}$). This class has a more complex structure; we give a partial characterization in terms of elementary graph properties, then a complete characterization by means of a (polynomial time) decision algorithm that accepts if and only if a robust MIS exists. This algorithm can be adapted to construct such a solution if one exists.

Keywords: Heredity in graphs, Highly-dynamic networks, Minimal independent sets, Temporal covering structure.

Email addresses: arnaud.casteigts@labri.fr (Arnaud Casteigts), swan.dubois@lip6.fr (Swan Dubois), franck.petit@lip6.fr (Franck Petit), robson@labri.fr (John M. Robson)

1. Introduction

The area of dynamic networks covers a variety of contexts, ranging from nearly-static networks where the network topology changes only occasionally, to highly-dynamic settings where the entities interact in a volatile way, through communication links which appear and disappear arbitrarily often and unexpectedly. In the second case, the immediate structure of the communication graph at given time (i.e., its structural snapshot) does not capture much information, the main features being rather of a temporal nature. For example, the snapshots may never be connected, and yet offer a form of connectivity over time and space, called *temporal connectivity*. A number of formalisms were proposed recently to capture the temporality of these contexts, such as evolving graphs, time-varying graphs, link streams, and temporal graphs (see e.g. [12, 8, 20, 18], among many others). In this article, we introduce a graph concept which is strongly motivated by the highly-dynamic setting; however, the notion itself can be formulated and studied in terms of standard graphs, independently from its temporal interpretation.

Given a (standard) graph G, a given property (or structure) in G is said to be *robust* if and only if it is inherited by every *connected spanning subgraph* of G. Hereditary properties based on the removal of edges are generally called *monotone* (see for instance [3, 4, 19]). Robustness is therefore a particular case of monotonicity, in which the subgraph is additionally constrained to remain spanning and connected. (This concept is different from other uses of the term "robustness" in the literature, see e.g. [6, 10, 14].)

As explained, robustness can be interpreted in several different ways. (1) *Static networks with permanent link crashes:* Here, the network is essentially static; however, it deteriorates over time and some of the links may definitively stop working (or equivalently, be removed). The network is to be used so long as it still connects all the nodes. It is easy to see, that if a property is robust in such a network, then it will remain valid so long as the network is used, despite the uncertainty regarding which of the links will crash. (2) *Recurrent temporal connectivity in highly-dynamic networks:* Here, the immediate structure of the network does not matter as much as its temporal properties. In particular, a basic assumption is that temporal paths exist recurrently between all the entities, corresponding to Class $\mathcal{TC}^{\mathcal{R}}$ in [7] and Class 5 in [8]. Dubois et al. observed in [11] that this assumption is equivalent to the guarantee that a subset of the edges always reappears, although in general such as subset is not known in advance. Thus, robustness can be interpreted in $\mathcal{TC}^{\mathcal{R}}$ as the fact that a property is satified with respect to the subset of recurrent edges, whichever these edges are. The choice for a given interpretation is not mandatory, as robustness can be studied independently. However, the second interpretation being the original motivation here, we develop it further in a dedicated (but optional) section in the end of the paper.

In this paper, we illustrate the concept of robustness through a classical covering problem called *maximal independent set* (MIS), which consists of selecting a subset of vertices none of which are neighbors (independence) and which is maximal for inclusion. Let us first observe that robust MISs may or may not exist depending on the considered graph. For example, if the graph is a triangle, then only one such structure exists up to isomorphism, consisting of a single vertex (refer to Figure 1a). If an edge next to the selected vertex is removed, then this set is no longer maximal. Therefore, the triangle graph admits no robust MIS. Some graphs admit both robust and non-robust MISs, as exemplified by the bull graphs on Figures 1b (non-robust) and 1c (robust). Finally, some graphs like the square graph (Figure 1d) are such that *all* MISs are robust.

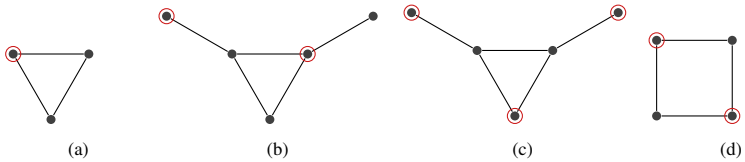

Figure 1: Four examples of MISs in various graphs.

1.1. Contributions

In addition to the concept itself and the related discussions, we characterize exactly the set $\mathcal{RMIS}^{\forall}$ of the graphs in which all MISs are robust. To this end, we first define a class of graphs called *sputniks* (for reasons that will become clear later). Sputniks include, among others, all the trees, for which every property is trivially robust (since none of the edges are removable). We show that $\mathcal{RMIS}^{\forall}$ consists exactly of the union of sputniks and complete bipartite graphs. The interest of this *universal* class is that finding a robust MIS in it amounts to finding a standard MIS.

The *existential* versions of this class, namely the set $\mathcal{RMIS}^{\exists}$ of those graphs which admit a robust solution seem to have a more complex structure. We first give a sufficient condition and we show that this condition is necessary in the particular case of biconnected graphs (meaning here 2-vertex-connected). The more general case is addressed by means of an algorithm that decides if a given graph belongs to $\mathcal{RMIS}^{\exists}$. The trivial strategy for such an algorithm would amount to enumerating all MISs until a robust one is found. However, exponentially many MISs may exist in general graphs (see Moon and Moser [22], and [13, 15] in the particular case of connected graphs) and the validity of each one may have to be satisfied in exponentially many connected spanning subgraphs. Motivated by this observation, we present a polynomial time decision algorithm, which can be adapted into a constructive algorithm (without significant overhead). Our algorithm relies on a particular decomposition of the graph into a tree of 2-vertex-connected (biconnected) components called an $\mathcal{ABC}$-tree (a variant of block-cut trees [16]), along which constraints are propagated as to the MIS status of intermediate vertices. The inner constraints of non-trivial components are solved by a reduction to the 2-SAT problem. The yes-instances of this algorithm characterize $\mathcal{RMIS}^{\exists}$, albeit indirectly. Whether $\mathcal{RMIS}^{\exists}$ admits a more elementary characterization in terms of graph properties is left open.

1.2. Organization of the paper

Section 2 presents the main definitions and concepts. Next, Section 3 presents the characterization of class $\mathcal{RMIS}^{\forall}$. In Section 4.1, we show that if the graph is biconnected, then being bipartite is both necessary and sufficient to belong to $\mathcal{RMIS}^{\exists}$. Then, we present the decision algorithm in Sections 4.2 through 4.6, its complexity in Section 4.7 and its adaptation into a constructive algorithm in Section 4.8. In Section 5, we develop the discussion regarding the temporal interpretation of robustness, motivated by highly-dynamic networks. We conclude in Section 6 by observing some additional features of robustness in general and a few open questions.

2. Main Concepts and Basic Results

Let $G = (V, E)$ be a simple connected undirected graph on a finite set V of n vertices (or nodes). We denote by $N(v)$ the neighbors of vertex v, i.e., the set $\{w \mid$

$\{v, w\} \in E\}$. The degree of a vertex v is $|N(v)|$. A vertex is *pendant* if it has degree 1. An *articulation point* (or *cut vertex*) is a vertex whose removal disconnects the graph. A *bridge* (or *cut edge*) is an edge whose removal disconnects the graph. We say that an edge is *removable* in a graph G if it is not a bridge of G. Given $V' \subseteq V$, the induced subgraph $G[V']$ is the graph whose vertex set is V' and whose edge set consists of all of the edges in E that have both endpoints in V'. In the context of this paper, a graph is said to be *biconnected* if it has at least three vertices, and it remains connected after the removal of any single vertex (i.e., 2-vertex-connectivity). A biconnected *component* is a maximal biconnected subgraph. Finally, a *spanning connected subgraph* of a graph $G = (V, E_G)$ is a graph $H = (V, E_H)$ such that $E_H \subseteq E_G$, and H is connected. We define the concept of robustness as follows.

Definition 1 (Robustness). *A property P is* robust *in G if and only if it is satisfied in every connected spanning subgraph of G (including G itself).*

In other words, a robust property holds even after an arbitrary number of edges are removed without disconnecting the graph. Robustness is a special case of a hereditary property, and more precisely a special case of a decreasing monotone property (see for instance [19]). The term "property" includes both basic graph properties and solutions to combinatorial problems. Our focus in this initial work is on the latter; however, looking at the robustness of basic graph properties might help understand this notion further, for instance, connectivity itself is a trivial robust property. Bipartiteness is also a robust that is discussed at several occasions in this paper.

In the present work, we focus on the *maximal independent sets* (MIS) problem, which consists of selecting a subset of vertices none of which are neighbors (independence) and to which no further vertex can be added (maximality). Following Definition 1, a robust MIS (RMIS, for short) in a graph G is a subset of vertices which remains *maximal* and *independent* in every connected spanning subgraph of G.

Observation 1. *The notion of independence is stable under the removal of edges. Therefore, it is sufficient that an MIS remains maximal in order to be an RMIS.*

Let us define the following two classes of graphs.

Definition 2 ($\mathcal{RMIS}^{\forall}$). *The set of graphs in which all MISs are robust.*

Definition 3 ($\mathcal{RMIS}^{\exists}$). *The set of graphs that admit at least one robust MIS.*

Trivially, $\mathcal{RMIS}^{\forall} \subseteq \mathcal{RMIS}^{\exists}$. Finally, we define two classes of graphs which play a central role in this work, namely *complete bipartite graphs* and *sputnik graphs*, the latter being new.

Definition 4 (Complete bipartite graph). *A complete bipartite graph is a graph $G = (V_1 \cup V_2, E)$ such that $V_1 \cap V_2 = \emptyset$ and $E = \{\{v_1, v_2\}, v_1 \in V_1$ and $v_2 \in V_2\}$.*

In other words, the vertices can be partitioned into two sets V_1 and V_2 such that every vertex in V_1 shares an edge with every vertex in V_2 (completeness), and these are the only edges (bipartiteness).

Definition 5 (Sputnik). *A graph is a* sputnik *if and only if every vertex v belonging to a cycle has at least one pendant neighbor.*

An example of a sputnik is shown on Figure 2. This name was chosen by analogy with the well-known satellite. By the same analogy, we say that a vertex has an *antenna* if it has at least one pendant neighbor.

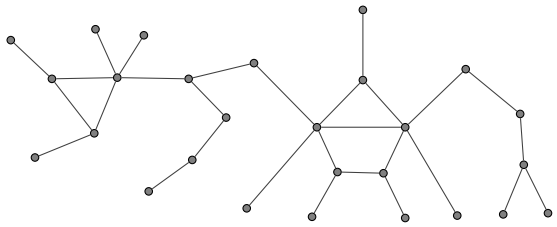

Figure 2: Example of a sputnik graph (whose planarity is accidental).

3. Characterization of $\mathcal{RMIS}^{\forall}$

In this section, we show that the class of graphs in which all MISs are robust, $\mathcal{RMIS}^{\forall}$, corresponds exactly to the union of complete bipartite graphs and sputnik graphs. We first show that all the MISs in a complete bipartite graph are robust, and so are all MISs in a sputnik graph (sufficient conditions). Next, we show that these graphs are the only ones (necessary condition). The necessary side is more complex because two classes of graphs are involved. The proof proceeds by showing that if all MISs are robust in a graph which is not a sputnik, then that graph *must* be a complete bipartite graph.

Lemma 1. *All MISs in a complete bipartite graph are robust.*

Proof: From Observation 1, we only need to show that maximality is preserved. There are two ways of chosing an MIS in a complete bipartite graph $G = (V_1 \cup V_2, E)$, namely V_1 or V_2. Without loss of generality, let V_1 be chosen. Then, in all connected spanning subgraphs of G, every vertex in V_2 still has at least one neighbor in V_1 (otherwise the graph is disconnected), which preserves maximality. □

Lemma 2. *All MISs in a sputnik graph are robust.*

Proof: Again, by Observation 1, we only need to show that maximality is preserved. By definition of a sputnik graph (Definition 5), if an edge $\{u, v\}$ is removable (i.e., it belongs to a cycle), then both of its endpoints have an antenna. Maximality implies that either u or the tip of its antenna are in the set. (The same holds for v.) As antennas are bridges, they cannot be removed and thus maximality is preserved when edges from a cycle are removed. □

Lemma 3. *If G is not a sputnik, and every MIS in G is robust, then G must be a complete bipartite graph.*

Proof: If G is not a sputnik, then some vertex u belonging to a cycle $C \subseteq G$ has no antenna. Consider the graph $G \setminus \{u\}$ and call $X_1, X_2, \ldots, X_k$ the components that would result, except that a copy of u is added back to each (see Figure 3 for an illustration). Without loss of generality, let X_1 be the one containing C. The other components (if any) are such that every neighbor of u has another neighbor than u (otherwise u would have a pendant neighbor).
Claim. If all MISs in G are robust, then all neighbors of u in X_1 have the same set of neighbors.

We prove this claim by contradiction. Let two neighbors v_1, v_2 of u be such that $N(v_1) \neq N(v_2)$. We will show that at least one MIS is not robust (see Figure 3 for an illustration). Without loss of generality, let x be a vertex in $N(v_1) \setminus N(v_2)$. Then an MIS can be built which contains both v_2 and x (as a special case, x may be the same vertex as v_2, which is not a problem). For each of the components $X_{i\geq 2}$, choose an edge $\{u, w_i\} \in X_i$ and add another neighbor of w_i to the MIS (such a neighbor exists, as we have already seen). One can see that u, v_1 and all w_i can no longer enter the MIS because they all have neighbors in it. Now, choose the remaining elements of the MIS arbitrarily. Now, consider the following removals: in all components $X_{i\geq 2}$ all edges incident to u except $\{u, w_i\}$ are removed; and in X_1, all edges incident to u except $\{u, v_1\}$ are removed. The resulting graph is connected since all of the $X_i \setminus \{u\}$ are connected, but the set is no longer maximal because u could now be added in it. *(End of Claim)*

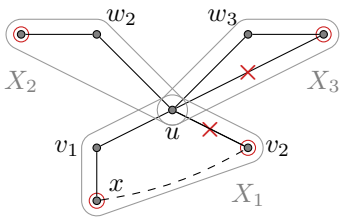

Figure 3: Contradictory example for proving the claim, where $x \in N(v_1) \setminus N(v_2)$. The dashed line represents the rest of component X_1.

Since u's neighbors in X_1 have the same neighbors, this means in particular that none of these vertices has an antenna. As a result, the arguments that applied to u because of its absence of antenna, apply in turn to u's neighbors in X_1. In particular, the neighbors of these vertices (including u) must have the same set of neighbors. This implies that (1) u can no longer be an articulation point, thus $G = X_1$ and (2) all neighbors of u have the same set of neighbors, and these neighbors also have the same set of neighbors, which finally implies that the graph is complete bipartite. □

Lemmas 1, 2, and 3 allow us to conclude with Theorem 1 :

Theorem 1. *All MISs are robust in a graph G if and only if G is complete bipartite or sputnik.*

4. Characterization of $\mathcal{RMIS}^{\exists}$

In this section, we turn our attention to the characterization of graphs which *admit* a robust MIS ($\mathcal{RMIS}^{\exists}$). Unfortunately, this class does not seem to admit a simple characterization in terms of elementary graph properties. We start by discussing some such properties that give a partial characterization, namely we show that bipartiteness is a necessary and sufficient condition for *biconnected* graphs to be in $\mathcal{RMIS}^{\exists}$. Then, we turn our attention to the general case and present an algorithm that decides if a given graph admits a robust MIS (in polynomial time, and constructively). The yes-instances of this algorithm are an indirect characterization of $\mathcal{RMIS}^{\exists}$.

4.1. Bipartiteness versus Biconnectivity

Let us first observe that bipartiteness is a sufficient condition for any graph to admit a robust MIS.

Lemma 4. *All bipartite graphs admit a robust MIS.*

Proof: The argument generalizes that of Lemma 1 for complete bipartite graphs. Let an MIS be composed of all the vertices of the first part. As long as the graph remains connected, every vertex in the second part has a neighbor in the first part, and thus in the MIS, which preserves maximality. (Independence is not impacted—Observation 1.) □

In fact, bipartiteness happens to be also a necessary condition in the particular case of biconnected graphs.

Lemma 5. *If a biconnected graph G is not bipartite, then it admits no robust MIS.*

Proof: By contradiction, suppose that G admits a robust MIS. As G is not bipartite, it is not 2-colorable, thus either two neighbor vertices exist which are both included in the set, or two neighbor vertices exist which are both excluded from the set. In the first case, independence is contradicted. In the second case, let u and v be two such vertices. Because the graph is biconnected, it is possible to remove all the incident edges to u except $\{u, v\}$ without disconnecting G, resulting in a non-maximal set, thus contradicting robustness. □

The argument in the proof of Lemma 5 will be used several times in the rest of the paper. We refer to it through the concept of a weak vertex.

Definition 6 (Weak vertex). *Let u be a vertex that is not included in an MIS. If u has another neighbor v not being included in the MIS and such that all edges incident to u except $\{u, v\}$ can be simultaneously removed without disconnecting the graph, then u is called a* weak *vertex.*

Lemmas 4 and 5 imply that the intersection of $\mathcal{RMIS}^{\exists}$ with biconnected graphs is *exactly* the set of bipartite biconnected graphs (see Table 1). An immediate corol-

	Bipartite	¬ Bipartite
Biconnected	yes	no
¬ Biconnected	yes	possibly

Table 1: Existence of a robust MIS with respect to bipartiteness and biconnectedness.

lary is the existence of an infinite family of graphs which do not admit a robust MIS, namely all biconnected non-bipartite graphs (including, for example, the triangle graph in Figure 1 and more generally all the odd cycles).

4.2. Overview of the algorithm

The problem of computing a *standard* MIS in a graph $G = (V_G, E_G)$ can be solved by a one-sentence greedy algorithm as follows: for all vertices $v \in V_G$ in arbitrary order, include v in the MIS if none of its neighbors already is. The problem of computing a *robust* MIS (or RMIS) is fundamentally different in two respects: (1) Solutions may or may not exist, and (2) Even if a solution exists, a decision made in some part of the graph can restrict (or invalidate) the feasible choices in remote parts. For example, in

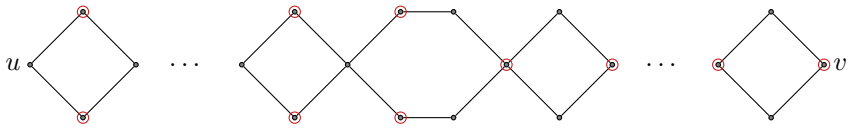

Figure 4: Non-locality of finding a robust MIS.

the graph of Figure 4, if vertex v is included in the set, then an RMIS exists if and only if node u is not included in the set. (Any other choice would produce a weak vertex.)

More generally, deciding whether a graph admits an RMIS requires the identification and propagation of constraints within the graph. To do so, our algorithm relies on a particular type of decomposition of the input graph as a tree of biconnected components (i.e.,, biconnected subgraph that are maximal) called $\mathcal{ABC}$-tree. The constraints are first determined at the leaves of this tree, then they are propagated and modified upward, until the root component is itself analysed. At an intermediate node of this tree, either the corresponding subtree admits an RMIS or it does not. If it does, a condition may apply regarding the status of the topmost vertex in the subtree, e.g. the subtree may admit an RMIS at the condition that this vertex is (or is not) in the MIS.

In the following, we always refer to the vertices of the input graph as *vertices*, and to those of the decomposition tree as *nodes* to avoid confusion. Note that if G is a tree, then none of its edges can be removed (thus all MISs are robust). As a result, we restrict our attention to the cases that G has at least one biconnected component. We also assume that G is connected, as otherwise each part can be solved separately.

4.3. The $\mathcal{ABC}$-tree decomposition

An $\mathcal{ABC}$-tree, denoted by $T = (V_T, E_T)$, is neither a block-cut tree, nor a bridge tree (see [16] for background), it is a mixture of both. Precisely, an $\mathcal{ABC}$-tree is made of four types of nodes $\mathcal{A}, \mathcal{B}, \mathcal{C}$ and $\mathcal{P}$ defined with respect to the input graph $G = (V_G, E_G)$ as follows:

- $\mathcal{P}$ is the set of pendant vertices,
- $\mathcal{A}$ is the set of articulation points,
- $\mathcal{B}$ is the set of bridges,
- $\mathcal{C}$ is the set of biconnected component.

Thus, every node in $\mathcal{P}$ and $\mathcal{A}$ corresponds to an original *vertex* in V_G (the converse is not true), every node in $\mathcal{B}$ corresponds to an *edge* in E_G (same remark), and every node in $\mathcal{C}$ corresponds to an entire *subgraph* of G (same remark again). Considering the graph in Figure 5, one would obtain $\mathcal{P} = \{4, 5, 7, 12, 20, 24\}$, $\mathcal{A} = \{2, 3, 6, 8, 10, 11, 14, 15, 16, 17, 18, 21, 22, 28\}$, $\mathcal{C} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}\}$, and $\mathcal{B}$ contains bridges such as $\{2, 3\}$, $\{3, 4\}$, $\{3, 5\}$, $\{6, 7\}$, $\{8, 14\}$, *etc.*

Observe that the same vertex of V_G can simultaneously be a node in $\mathcal{A}$, the endpoint of one or several bridges in $\mathcal{B}$, and a node belonging to one or several biconnected components in $\mathcal{C}$. Vertices 2, 6, 8, and 10 are examples of such vertices. Also observe that the endpoints of a bridge are always articulation points or pendant vertices. All these relations are materialized by the set of edges E_T of the $\mathcal{ABC}$-tree, defined as

$$E_T = \{\{a, X\} \in \mathcal{A} \times \mathcal{C} \mid a \in X\} \cup \{\{v, Y\} \in (\mathcal{A} \cup \mathcal{P}) \times \mathcal{B} \mid v \in Y\}.$$

In words, articulation points (seen as nodes) share an edge with the biconnected components they belong to (if any), and articulation points or pendant nodes (seen as nodes) share an edge with the bridges they belong to (if any). Observe that the resulting graph (the $\mathcal{ABC}$-tree) is indeed acyclic, as otherwise two distinct paths would exist between distinct biconnected components, which contradicts the fact that these components are maximal. The $\mathcal{ABC}$-tree corresponding to the graph of Figure 5 is shown in Figure 6. The reader is encouraged to spend a few minutes getting acquainted with this construction, which is used frequently in the following.

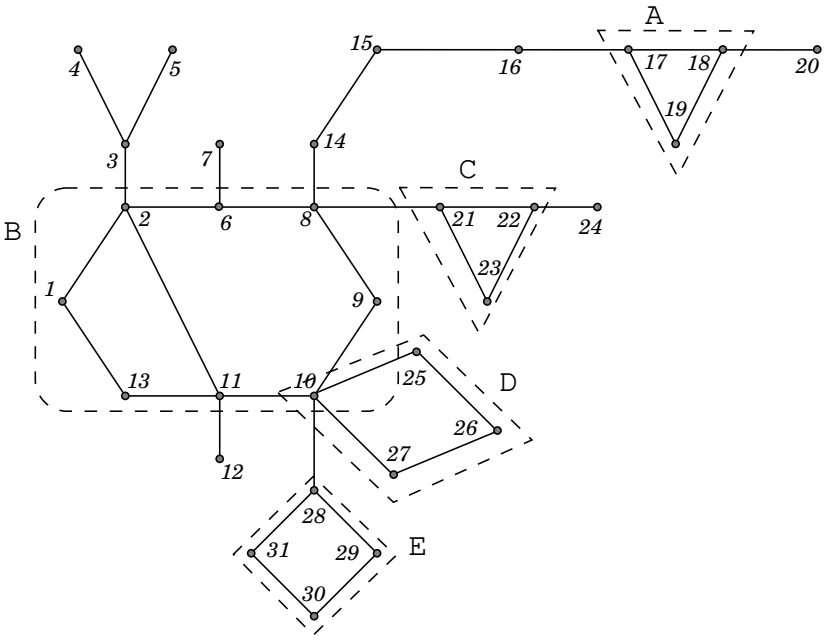

Figure 5: Example of input graph, seen as a tree of biconnected components.

4.4. RMIS constraints in a rooted $\mathcal{ABC}$-tree

The algorithm proceeds by propagating constraints from the leaves of the $\mathcal{ABC}$-tree to a root R chosen arbitrarily among the biconnected components. Given a node in $x \in V_T \setminus R$, we denote by $\texttt{parent}(x)$ the neighbor of $x \in V_T$ that is one-hop closer to R, and by $\texttt{children}(x)$ the set of nodes $\{y \in V_T \setminus R \mid \texttt{parent}(y) = x\}$. We extend these definitions to $\texttt{descendants}()$ in the natural way. Because some nodes in T correspond to real vertices (namely, the nodes in $\mathcal{A}$ and $\mathcal{P}$) and others do not (the nodes in $\mathcal{B}$ and $\mathcal{C}$), we define a notion of *attachment vertex* $v(x)$ of a node x as being either the underlying vertex itself (if $x \in \mathcal{A} \cup \mathcal{P}$) or the underlying vertex of the parent (if $x \in \mathcal{B} \cup \mathcal{C}$). Indeed, the parent of a node in $\mathcal{B} \cup \mathcal{C}$ is always a node in $\mathcal{A}$, thus it corresponds to a real vertex in V_G. Intuitively, the attachment vertex of a node is the highest vertex in this node towards the root R. For instance, in Figure 6, assuming that R is component $\texttt{D}$, then we have for instance $v(4) = 4$, $v(\{3,4\}) = 3$, $v(3) = 3$, $v(\{8,14\}) = v(\{8,21\}) = v(8) = 8$, and $v(\{10,28\}) = v(\texttt{B}) = v(10) = 10$, and indeed, 10 is the highest underlying vertex in all these nodes (with respect to the root).

Attachment vertices play a important role in the management of constraints, because the constraints induced by x are ultimately aggregated as a membership status

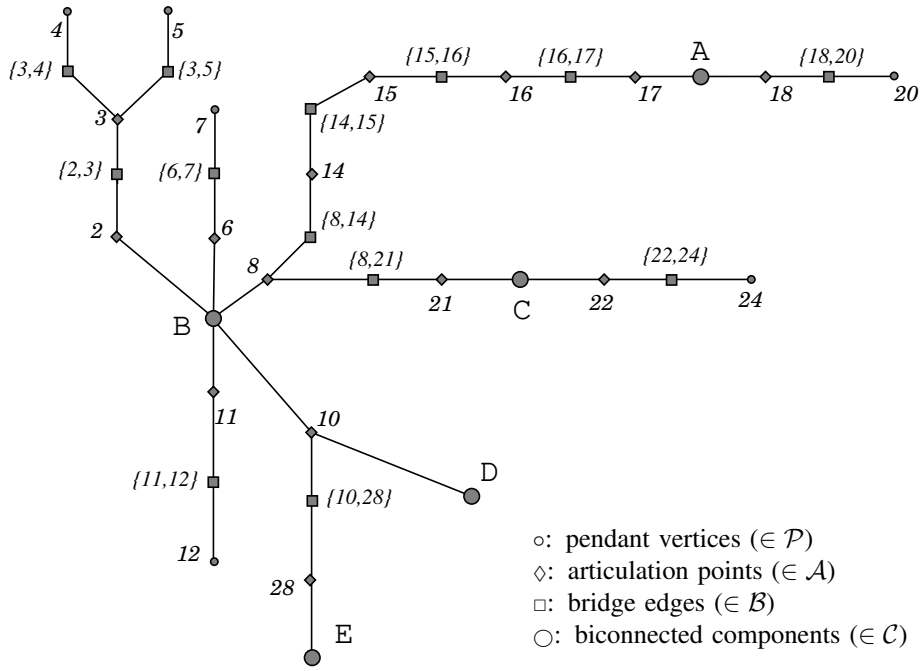

Figure 6: $\mathcal{ABC}$-tree corresponding to the input graph on Figure 5.

for $v(x)$ in the MIS. The more formal definition relies on induced subgraphs as follows. Every node x induces a subgraph $G_x \subseteq G$ which is exactly $G[x]$ when $x \in \mathcal{A} \cup \mathcal{B} \cup \mathcal{P}$ (see definitions in Section 2), or x itself when $x \in \mathcal{C}$. By extension, for any $x \in V_T$, the subtree $T_x = (V_{T_x}, E_{T_x}) \subseteq T$ whose highest node with respect to R is x induces a subgraph $G[T_x] \subseteq G$ defined as $\cup G_y$ over all $y \in (\texttt{descendants}(x) \cup x)$. In other words, $G[T_x]$ is the subgraph of G which corresponds to the subtree of x. We define the concept of *aerial* version of $G[T_x]$, noted $G_a[T_x]$ as $G[T_x] \cup G[\{v(x), a\}]$ with a an artificial neighbor of $v(x)$ that does not belong to $G[T_x]$. For any x, the constraints applying to $v(x)$ are encoded using the three following labels (also called tags):

- *PI* (Possibly In): $G[T_x]$ admits an RMIS that includes $v(x)$;
- *PO* (Possibly Out): $G[T_x]$ admits an RMIS that does not include $v(x)$;
- *PE* (Possibly External): $G_a[T_x]$ admits an RMIS that does not include $v(x)$.

Observe that a node may have several labels at the same time. For example *PI* and *PO*, or *PI* and *PE*. On the other hand, we use *PO* and *PE* in a mutually exclusive way based on the following remark.

Remark 1. *If a node x has label PO, then $G[T_x]$ admits an RMIS in which $v(x)$ is not included but one of its neighbor in $G[T_x]$ is (due to maximality). Thus, no inclusion constraint applies regarding the external neighbor of $v(x)$. As a result, whenever the constraints from different children induce both PO and PE, PO is chosen.*

4.5. Constraint identification and propagation

In this subsection, we present the rules used for identifying and propagating constraints within the $\mathcal{ABC}$-tree. The purpose of the rules is to determine what labels $L(x)$

a node x should take based on the labels of its children in the $\mathcal{ABC}$-tree. The validity of the rules is established gradually along their descriptions, based on the following definition of a correct labeling.

Definition 7 (Correct labeling). *A node x (in the underlying rooted $\mathcal{ABC}$-tree T) is correctly labeled if*

- *PI $\in L(x)$ if and only if $G[T_x]$ admits an RMIS that includes $v(x)$;*
- *PO $\in L(x)$ if and only if $G[T_x]$ admits an RMIS that does not include $v(x)$; and*
- *PE $\in L(x)$ if and only if PO $\notin L(x)$ and $G_a[T_x]$ admits an RMIS that does not include $v(x)$.*

The rules are presented below based on the type of x (namely, $\mathcal{A}, \mathcal{B}, \mathcal{C}$, and $\mathcal{P}$). They are illustrated in reference to the input graph $G = (V_G, E_G)$ in Figure 5 and the corresponding $\mathcal{ABC}$-tree $T = (V_T, E_T)$ in Figure 6, with root $\mathtt{D} \in \mathcal{C}$. For simplicity, when discussing about the construction of an MIS, if a vertex v is not included in the MIS and none of its neighbors is included, we say that v is *not covered*. The other vertices are *covered*. (This terminology is standard in the literature on covering problems.)

4.5.1. Pendant nodes ($x \in \mathcal{P}$)

This case is the easiest, because nodes in $\mathcal{P}$ have no children and their labels are always the same.

Labeling rule 1. *$L(x)$ is set to $\{$PI, PE$\}$.*

Labeling rule 1 is given by Lines 03 to 04 in Algorithm 1. In words, $v(x)$ may or may not be included to the MIS, but if it is not, then $v(y)$ should be, where y is the bridge node such that $\mathtt{parent}(x) = y$.

Lemma 6. *If $x \in \mathcal{P}$, then Labeling rule 1 produces a correct labeling of x.*

Proof: If $v(x)$ is included in the solution set, then this set (made of $v(x)$ alone) is clearly maximal and independent in $G[T_x]$ (which is also $v(x)$ alone), thus x can be labeled *PI*. If it is not included, but $v(\mathtt{parent}(x))$ is included, then the resulting set is maximal and independent in $G_a[T_x]$ (i.e., the graph made of a single edge between x and $\mathtt{parent}(x)$), thus x can be labeled *PE*. In both cases, the considered subgraph of G is itself a tree, thus any valid MIS in it is robust, thus labels *PI* and *PE* are both valid. On the other hand, x cannot be labeled *PO* because if $v(x)$ is not in the MIS, then the corresponding set is not maximal in $G[T_x]$ (which is reduced to the single vertex $v(x)$). □

4.5.2. Articulation points ($x \in \mathcal{A}$)

By construction of the $\mathcal{ABC}$-tree, if $x \in A$, then all the children of x are in $\mathcal{B}$ or in $\mathcal{C}$ and their attachment vertex is nothing but $v(x)$ itself. Thus, the constraints of each set of children already relates to $v(x)$, albeit individually. The rule consists of aggregating these constraints as follows.

Labeling rule 2. *If PI belongs to the labels of all the children of x, then PI is added to $L(x)$; if all the children of x contain a label PE or PO, then two subcases arise: either none of them contains PO, in which case PE is added to $L(x)$, or at least one contains PO, in which case PO is added to $L(x)$.*

The formal description of Labeling rule 2 is given by Lines 6 to 13 in Algorithm 1 on page 14 (the algorithm itself is discussed in a subsequent section). In our example, nodes 3, 11, and 18 are all labeled *PI* and *PO*.

Lemma 7. *If $x \in \mathcal{A}$ and all the nodes in $\texttt{children}(x)$ are correctly labeled, then Labeling rule 2 produces a correct labeling of x.*

Proof: Let us assume that all the nodes in $\texttt{children}(x)$ are correctly labeled. The proof follows the same cases as the labeling rule. We first prove that the assigned labels are valid, then we prove that they cannot be assigned otherwise.

- If all the nodes in $\texttt{children}(x)$ contain label *PI*, then for all $y \in \texttt{children}(x)$, $G[T_y]$ admits a robust MIS that includes $v(y)$. But since $x \in \mathcal{A}$, we have $v(y) = v(x)$, thus $G[T_x]$ admits a robust MIS that includes $v(x)$.
- If all the nodes in $\texttt{children}(x)$ contain label *PE* or *PO*, then two possible subcases arise:
  1. None of them contains *PO*. In this case, all of them contain label *PE*, meaning that for all $y \in \texttt{children}(x), G[T_y]$ does not admit a robust MIS that excludes $v(y)$, but $G_a[T_y]$ does. Since $v(x) = v(y)$, we have that $G[T_x]$ does not admit a robust MIS that excludes $v(x)$ (a single child with label *PE* would actually be enough here), but $G_a[T_x]$ does (here, we need that *all* the children have label *PE*).
  2. At least one contains *PO*. In this case, at least one child y is such that $G[T_y]$ admits an RMIS that excludes $v(y)$. Since $v(y) = v(x)$, at least one of the neighbors of $v(x)$ in $G[T_y]$ can be included in such an MIS, which satisfies the aerial constraints of the $G_a[T_{y'}]$ for all other children y' labeled *PE* (if any). Thus $G[T_x]$ admits an RMIS that excludes $v(x)$ without requiring further aerial constraints above x.

We focus now on the negative direction. If $G[T_x]$ does not admit a robust MIS that includes $v(x)$, then because $v(x) = v(y)$, there exists at least one $y \in \texttt{children}(x)$ such that $G[T_y]$ does not admit a robust MIS that includes $v(y)$. As the labeling of y is correct, y is not labeled *PI* and hence $v(x)$ is not labeled *PI* by the rule. Similarly, if $G[T_x]$ does not admit a robust MIS that excludes $v(x)$, then because $v(x) = v(y)$, the two cases above are possible. (i) There exists at least one $y \in \texttt{children}(x)$ such that neither $G[T_y]$ nor $G_a[T_y]$ admit a robust MIS that excludes $v(y)$. As the labeling of y is correct, y is not labeled *PO* or *PE*. (ii) For any $y \in \texttt{children}(x)$, $G[T_y]$ does not admit a robust MIS that excludes $v(y)$. As the labeling of y is correct, y is not labeled *PO*. In both cases, $v(x)$ cannot be labeled *PO* by the rule. Finally, if $G_a[T_x]$ does not admit a robust MIS that excludes $v(x)$, then because $v(x) = v(y)$, there exists at least one $y \in \texttt{children}(x)$ such that $G_a[T_y]$ does not admit a robust MIS that excludes $v(y)$. As the labeling of y is correct, y is not labeled *PE* and hence $v(x)$ is not labeled *PE* by the rule. □

4.5.3. Bridge nodes ($x \in \mathcal{B}$)

If x is a bridge node, then it has exactly one child y whose attachment vertex $v(y)$ is a neighbor of $v(x)$. The rule consists of transforming the existing constraints on $v(y)$ into constraints on $v(x)$ as follows.

Labeling rule 3. *If $L(y)$ contains label PI, then label PO is added to $L(x)$; if $L(y)$ contains label PE, then PI is added to $L(x)$; if $L(y)$ contains label PO, then PI and PE are added to $L(x)$. Finally (cleaning), if both PE and PO have been added to $L(x)$, then PE is removed from $L(x)$ (see Remark 1).*

The formal description of Labeling rule 3 is given by Lines 15 to 23 in Algorithm 1. In our example, the nodes $\{3, 4\}$, $\{3, 5\}$, and $\{18, 20\}$ are all tagged *PI* and *PO*.

Lemma 8. *If $x \in \mathcal{B}$ and y is correctly labeled, then Labeling rule 3 produces a correct labeling of x.*

Proof: Let $x \in \mathcal{B}$ and $\texttt{children}(x) = \{y\}$.

- If $L(y)$ contains label *PI*, then $G[T_y]$ admits a robust MIS including $v(y)$. Because $v(y)$ is included, the MIS is also valid in $G[T_x] = G[T_y] \cup G[\{x, y\}]$, and because this edge is a bridge (i.e., it is not removable), it remains robust in $G[T_x]$, thus x can be labeled *PO*.
- If $L(y)$ contains label *PE*, then $G_a[T_y] = G[T_x]$ admits a robust MIS including $v(x)$, thus x can be labeled *PI*.
- If $L(y)$ contains label PO, then $G[T_y]$ admits a robust MIS that excludes $v(y)$.
  - As $v(y)$ is the only neighbor of $v(x)$ in $G[T_x]$, adding $v(x)$ to such an MIS would produce a valid MIS in $G[T_x] = G[T_y] \cup G[\{x, y\}]$, and because $\{x, y\}$ is a bridge, the resulting MIS would remain robust. Thus x can be labeled *PI*.
  - As $v(y)$ already has a neighbor included in the MIS in $G[T_y]$, not including $v(x)$ in the MIS means that $v(x)$ is the only uncovered vertex in $G[T_x]$, which can be remedied by including to the MIS an aerial neighbor in $G_a[T_x]$. Thus, x can be labeled *PE*.

We focus now on the negative direction. If $G[T_x]$ does not admit a robust MIS that includes $v(x)$, then because $\{x, y\}$ is not a removable edge, $G[T_y]$ does not admit a robust MIS that excludes $v(y)$ and $G_a[T_y] = G[T_x]$ does not admit a robust MIS that excludes $v(y)$ (and includes a). As the labeling of y is correct, y is not labeled *PI* or *PE* and hence $v(x)$ is not labeled *PI* by the rule. Similarly, if $G[T_x]$ does not admit a robust MIS that excludes $v(x)$, then because $\{x, y\}$ is not a removable edge, $G[T_y]$ does not admit a robust MIS that includes $v(y)$. As the labeling of y is correct, y is not labeled *PI* and hence $v(x)$ cannot be labeled *PO*. Finally, if $G_a[T_x]$ does not admit a robust MIS that excludes $v(x)$, then because $\{x, y\}$ is not a removable edge, $G[T_y]$ does not admit a robust MIS that excludes $v(y)$. As the labeling of y is correct, y is not labeled *PO* and hence $v(x)$ is not labeled *PE* by the rule. □

4.5.4. Biconnected components ($x \in \mathcal{C}$)

As discussed in Section 4.1, when G itself is a biconnected component, an RMIS exists if and only if G is bipartite (Lemmas 4 and 5). When a biconnected component x is a node in the $\mathcal{ABC}$-tree T, the situation is more complex due to the existence of

Algorithm 1 LabelNode(x)

Parameters: A node x whose children in T are already labeled
Return: A correct labeling of x

$L(x) \leftarrow \emptyset$

case $x \in \mathcal{P}$:
    $L(x) \leftarrow \{PI, PE\}$

case $x \in \mathcal{A}$:
    if $\forall c \in \texttt{children}(x), PI \in L(c)$ **then**
        $L(x) \leftarrow L(x) \cup PI$
    if $\forall c \in \texttt{children}(x), PO \in L(c)$ or $PE \in L(c)$ **then**
        if $\exists c \in \texttt{children}(x), PO \in L(c)$ **then**
            $L(x) \leftarrow L(x) \cup PO$
        else
            $L(x) \leftarrow L(x) \cup PE$

case $x \in \mathcal{B}$: (with child side c)
    if $PI \in L(c)$ **then**
        $L(x) \leftarrow L(x) \cup PO$
    if $PE \in L(c)$ **then**
        $L(x) \leftarrow L(x) \cup PI$
    if $PO \in L(c)$ **then**
        $L(x) \leftarrow L(x) \cup \{PI, PE\}$
    if $PO \in L(x)$ and $PE \in L(x)$ **then**
        $L(x) \leftarrow L(x) \setminus PE$

case $x \in \mathcal{C}$:
    if isSatisfiable$(x, \texttt{in})$ **then**
        $L(x) \leftarrow L(x) \cup PI$
    if isSatisfiable$(x, \texttt{out})$ **then**
        $L(x) \leftarrow L(x) \cup PO$
    else
        $L(\texttt{parent}(x)) \leftarrow PO$
        if isSatisfiable$(x, \texttt{out})$ **then**
            $L(x) \leftarrow L(x) \cup PE$
        $L(\texttt{parent}(x)) \leftarrow \emptyset$

return $L(x)$

constraints from neighbors in T. In particular, an RMIS satisfying these constraints may or may not exist.

Let G_x be the subgraph of G induced by node x (in fact, coinciding with x). By construction of the $\mathcal{ABC}$-tree, every neighbor of x in T (whether children or parent) corresponds to an articulation point y in $\mathcal{A}$ such that $v(y) \in G_x$. Thus, potential constraints on these nodes can be seen as applying to vertices *inside* G_x. This nice feature allows us to focus on finding an RMIS in G_x without worrying about the entire subtree $G[T_x]$. Indeed, if an RMIS M satisfying the constraints exists in G_x and if every child y is correctly labelled, then an RMIS M_y must exist in every $G[T_y]$ such that $v(y)$ has the same status (included or excluded) in M_y and in M. As every y is an articulation point, the union of all these RMISs forms a robust MIS in $G[T_x]$.

Based on the above observation, we now focus on the simpler problem of finding an RMIS in G_x that satisfies potential constraints on its articulation points. In addition, we add an input parameter to the problem that forces the status of the attachment vertex $v(x)$ in the RMIS, so that (for instance) x receives label *PI* if an RMIS is found that includes $v(x)$. (The exact labeling rules used by our algorithm is described in Subsection 4.5.4.2.)

4.5.4.1 Solving G_x.

Here, we consider the following problem. Let G_x be a biconnected component in which some vertices are articulation points in G (corresponding to nodes in $\mathcal{A}$). Let L be the labeling function that gives constraints on these articulations point. Let p be a parameter `in`, `out`, or `none`. The goal is to determine if there exists a robust MIS in G_x that satisfies the two following constraints:

1. $v(x)$ is included in the RMIS if the parameter is `in`, $v(x)$ is not included if the parameter is `out`, and $v(x)$ is unconstrained if the parameter is `none`;
2. the inclusion status of every constrained articulation point y matches at least one of its labels.

Namely, if $L(y) \neq \emptyset$ (i.e., y is constrained), then $v(y)$ can be included to the RMIS only if $L(y)$ contains *PI*; and it can be excluded from the RMIS only if $L(y)$ contains *PO* or *PE*.

Bipartiteness vs. PO labels. As already explained, if a biconnected graph is not bipartite, then weak vertices must exist (see Lemma 5 and Definition 6). Bipartiteness still plays a central role in the case of biconnected components, but the existence of *PO* labels makes it more subtle, as explained in the following remark.

Remark 2. *Let y be an articulation point such that $v(y) \in G_x$ and y contains label PO, then there exists an RMIS in $G[T_y]$ that does not include $v(y)$ itself but includes one (or possibly several) neighbors in $G[T_y]$ that prevent $v(y)$ from being a weak vertex.*

As a result, the vertices admitting a label *PO* are not subject to the same bipartiteness constraint as the others vertices.

The procedure. The procedure is formally described in Algorithm 2, to which the reader is referred for details, and its correctness is proved subsequently. Let E^{PO} be the set of edges in G_x such that both endpoints contain label *PO*, and let X be the subgraph $G_x \setminus E^{PO}$, possibly resulting into several connected components $X_1, ..., X_k$. If X is not bipartite, then the algorithm rejects, because this means that at least one

Algorithm 2 isSatisfiable$(x, flag)$

Parameters: A node $x \in \mathcal{C}$ and a $flag \in \{\texttt{in}, \texttt{out}, \texttt{none}\}$ that forces the status of $v(x)$
Return: `true` if a suitable set of x exists, `false` otherwise

Let $E^{\texttt{PO}}$ be the set of edges $\{u, v\}$ of G_x such that *PO* $\in L(u)$ and *PO* $\in L(v)$
Let $X = G_x \setminus E^{\texttt{PO}}$
if X is not bipartite **then**
    return false
Let $X_1, \ldots, X_k$ be the (bipartite) connected components in X
Let F be an empty 2-SAT expression on a set $\{x_1, \ldots, x_k\}$ of boolean variables
foreach i in $1..k$ **do**
    Assign to each vertex v in X_i a label $\ell(v) \in \{x_i, \neg x_i\}$ such that neighbors have different labels
foreach $a \in \mathcal{A} | v(a) \in G_x$ **do**
    if $L(a) = \{$*PI*$\}$ **then**
        $F \leftarrow F \wedge (\ell(v(a)))$
    if $L(a) = \{$*PO*$\}$ or $L(a) = \{$*PE*$\}$ **then**
        $F \leftarrow F \wedge (\neg\ell(v(a)))$
foreach $\{u, v\} \in E^{\texttt{PO}}$ **do**
    $F \leftarrow F \wedge (\neg\ell(u) \vee \neg\ell(v))$
if $flag = \texttt{in}$ **then**
    $F \leftarrow F \wedge (\ell(v(x)))$
else if $flag = \texttt{out}$ **then**
    $F \leftarrow F \wedge (\neg\ell(v(x)))$
return whether F is satisfiable (using an external 2-SAT solver)

non-PO vertex must be *weak*. Otherwise, the component may possibly admit an RMIS if the combination of constraints induced by all the vertices is satisfiable. The rest of the procedure consists of encoding these constraints into a 2-SAT formula such that G_x admits an RMIS (with given status for $v(x)$, the attachment vertex), if and only if the formula is satisfiable. The formula is built as follows. For each connected component X_i, a SAT variable x_i is created. As X_i is bipartite, every vertex v of one part receives label $\ell(v) \leftarrow x_i$ and every vertex v of the other part receives a label $\ell(v) \leftarrow \neg x_i$. (Intuitively, if the eventual formula is satisfiable with $x_i = true$, then the vertices in the first part are included in the RMIS; if it is satisfiable with $x_i = false$, then the vertices of the second part are included.) Then, the existing constraints of articulation points are incorporated; namely, if an articulation point $y \in \mathcal{A}$ contains *only* label *PI*, then $\ell(v(y))$ is set to `true`; if it contains only *PO* or only *PE*, then $\ell(v(y))$ is set to `false`; if it contains both options (i.e., *PI PO* or *PI PE*), then no constraint is added. Finally, although the edges of E^{PO} induce no constraints for bipartiteness, we must still make sure that their endpoints are not both included to the RMIS (for independence), thus if $\{u, v\} \in E^{PO}$, then the clause $\neg\ell(u) \vee \neg\ell(v)$ is added.

Correctness. To formalize the properties of the procedure `isSatisfiable`, we need to introduce some definitions. Let G_x^+ be the graph built from G_x by adding a small gadget to every $v(y) \in G_x$ when y is constrained (i.e., $L(y)$ is non-empty). The gadget consists of a path of length 2 incident to $v(y)$ through virtual vertices $v'(y)$ and $v''(y)$ (see Figure 7 for an intuition). This construction is not used in the procedure itself, only in the proof. We say that a set of vertices S is a *suitable RMIS* of G_x^+ if and only if S is an RMIS of G_x^+ such that, for every $y \in G_x$ with at least one label in $L(y)$, we have:
- $v(y)$ in S only if *PI* $\in L(y)$;
- $v(y)$ not in S only if *PO* or *PE* appears in $L(y)$; and
- $v(y)$ not in S, $v'(y)$ in S, and $v''(y)$ not in S only if PO appears in $L(y)$.

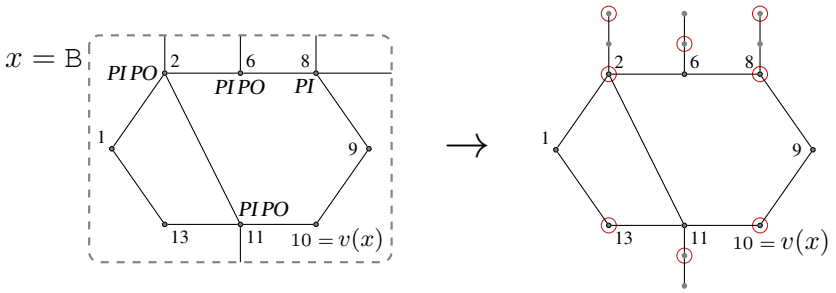

Figure 7: Example of component G_x (left)–x corresponds to the component B in Figure 5. With $v(x) = 10$, a chain of two virtual nodes is added to each constrained node $v(y) \in \{2, 6, 8, 11\}$ that gives the corresponding G_x^+ (right). The MIS shown on G_x^+ corresponds to the set S used in the proof of Lemma 9 with $v(x)$ included.

The idea is that, for every y, the graph induced by the path $\{v(y), v'(y), v''(y)\}$ replaces $G[T_y]$ in the management of constraints so that a suitable RMIS of G_x^+ exists if and only if an RMIS of $G[T_x]$ exists. We now prove the main lemma.

Lemma 9. *Given a biconnected component $x \in \mathcal{C}$ with constraints on some articulation points,* $\texttt{isSatisfiable}(x, \texttt{in})$ *returns true if and only if a suitable RMIS of G_x^+ including $v(x)$ exist;* $\texttt{isSatisfiable}(x, \texttt{out})$ *returns true if and only if a suitable RMIS of G_x^+ excluding $v(x)$ exist;* $\texttt{isSatisfiable}(x, \texttt{none})$ *returns true if and only if a suitable RMIS of G_x^+ exists, irrespective of $v(x)$.*

Proof: Let x be the considered node in the $\mathcal{ABC}$-tree and G_x the corresponding component. Let G_x^+ be the augmented version of G_x as described above. First observe that, if $G_x \setminus E^{PO}$ is bipartite, then so is $G_x^+ \setminus E^{PO}$, because the adjunction of such paths cannot affect bipartiteness. In the following, we first prove the direction that (1) if the procedure returns true, then the corresponding type of suitable RMIS exists; then (2) if the given type of suitable RMIS exists, then the procedure with corresponding parameters must return true.

(1) If F is satisfiable, define S as the set of vertices induced by a positive assignment, i.e., $\{v \in G_x | \ell(v) = true\}$. Now, add to S some of the virtual vertices as follows: if $v(y)$ is in S, add only $v''(y)$; if $v(y)$ is not in S and is weak in G_x, add only $v'(y)$; if $v(y)$ is not in S and is not weak in G_x, add only $v''(y)$. Lines 16 to 19 ensure that $v(x)$ is included in the MIS if $flag = \texttt{in}$ and that it is excluded if $flag = \texttt{out}$ (if $flag = \texttt{none}$, no constraint applies). We now prove that S is independent, maximal, robust, and suitable in G^+.

Independence: Let $e = \{u, v\}$ be an edge of G_x^+. If e is an edge from one of the extra paths, u and v cannot be both in S by construction. Otherwise (i.e., e is in G_x), either e belongs to E^{PO}, or it does not. If it does, then the clause introduced on Line 15 ensures that u and v cannot be both in the MIS. If it does not, then it belongs to $G_x^+ \setminus E^{PO}$, which is bipartite, then Line 08 (and the greedy process for extra paths) ensures that u and v cannot be both in the MIS.

Maximality and robustness: Recall that independence is not affected by the removal of edges (Observation 1), thus robustness means preserving maximality. Let u be a vertex of G_x^+ that does not belong to S. Either u belongs to an edge in E^{PO} or it does not. If it does, then by Remark 2 it has a neighbor in the MIS (maximality) and

it cannot be a weak vertex (robustness). If it does not, then *all* of its neighbors in G_x^+ are in the set because $G_x \setminus E^{PO}$ is bipartite (and the extension of S in the extra paths keep alternating the inclusion status), thus so long as G_x^+ remains connected, u cannot be added to S (maximality and robustness).

Suitability: For every $v(y) \in G_x$ such that $L(y) \neq \emptyset$. Lines 12 and 13 ensure that $v(y) \in S$ only if *PI* appears in $L(y)$. Lines 10 and 11 ensure that $v(y) \notin S$ only if *PO* or *PE* appear in $L(y)$. If $v'(y) \in S$ and $v''(y) \notin S$, then by construction $v(y) \notin S$ and it is weak in G_x. Line 08 implies that at least one adjacent edge to $v(y)$ in G_x is in E^{PO} and then *PE* does not appear in $L(y)$, implying that PO does.

(2) Let M be a suitable RMIS of G_x^+ in which the status of $v(x)$ is `in` (resp `out`). Let $X = G_x \setminus E^{PO}$. If X is not bipartite, then it contains a weak vertex, which contradicts the robustness of M. Thus, X must be bipartite. In general, X may be made of several connected components $X_1, ..., X_k$. A satisfying assignment can then be constructed from M as follows. For all i ranging from 1 to k, chose the value of x_i so that $\forall v \in G_x, \ell(v) = true \Leftrightarrow v \in M$. This assignment satisfies the mutual exclusivity constraint in Line 15 (with respect to edges in E^{PO}), as otherwise M would not be independent, and it also satisfies the constraint added with respect to $L(y)$ (Lines 09 to 13), by suitability of M. Finally, due to Lines 17 and 19, the assignment must include (respectively exclude) $v(x)$ if the input flag is `in` (resp. `out`). Thus a satisfiable assignment corresponding to M must exists. □

4.5.4.2 The Labeling Rule.

Let us now return to the labeling of node $x \in \mathcal{C}$ in the $\mathcal{ABC}$-tree, assuming that all y in $\texttt{children}(x)$ are correctly labeled. The goal here is to label x as per the possibilities, namely to assign label *PI* if an RMIS exists in $G[T_x]$ that includes $v(x)$; to assign label *PO* if an RMIS exists in $G[T_x]$ that excludes $v(x)$; and, in case the latter does not, to assign *PE* if an RMIS exists that excludes $v(x)$ provided that an external neighbor of it is subsequently included.

We need to distinguish here between the case that x is the root component R of the $\mathcal{ABC}$-tree, or just an internal node. If x is the root, then it has no attachment point and the only thing that matters is whether an RMIS exists or not. As such, x does not receive a labeling, and is instead treated directly from the main algorithm (described in the next section). Thus, we focus on how the above procedure can be used to label x when x is an internal node of the $\mathcal{ABC}$-tree.

The first two tests are realized by fixing the status of $v(x)$ as a parameter when calling `isSatisfiable`(). A first call is made, setting this parameter to `in`, then a second call is made, setting this parameter to `out`. If the second call fails (no such RMIS exist), then a third call is made with a different strategy. This strategy relies on a trick using label *PO*. Remark 2 implies that whenever a child y of x is labeled *PO*, then $v(y)$ (itself in x) does not need to have a neighbor *within* x that is included in the RMIS. As a result, testing whether $G[T_x]$ admits an RMIS if an external neighbor of $v(x)$ is subsequently added (i.e., label *PE*) can be done by pretending that the parent z of x, whose attachment vertex $v(z) = v(x)$ is in x, itself has an external neighbor in the MIS. Thus, just like the other children labeled *PO*, an artificial *PO* label is assigned to $\texttt{parent}(x)$, resulting in the forced non-selection of $v(x)$ to the MIS and the adjunction of an extra path to $v(x)$ (whose neighbor in the path *will* be included). The existence of

an RMIS satisfying this configuration implies that x can be labeled *PE*. The resulting labeling rule is as follows.

Labeling rule 4. *If* `isSatisfiable()` *returns true with parameter* `in`*, then PI is added to* $L(x)$*. Next, if* `isSatisfiable()` *returns true with parameter* `out`*, then PO is added to* $L(x)$*. Next, if PO* $\notin L(x)$ *and* `isSatisfiable()` *returns true with parameter* `out` *with an artificial PO label in* $L(parent(x))$*, then PE is added to* $L(x)$*.*

The corresponding instructions are formally given in Lines 25 to 34 of Algorithm 1. In our example (Figure 6), component `E` is labeled both *PI* (with the suitable set $\{28, 30\}$) and *PO* (with the suitable set $\{29, 31\}$) while `B` is labeled *PI* (with the suitable set $\{2, 8, 10, 13\}$) and *PE* (with the suitable set $\{1, 8, 11\}$).

Lemma 10. *If* $x \in \mathcal{C} \setminus \{R\}$ *and all nodes of* $children(x)$ *are correctly labeled, then Labeling rule 4 produces a correct labeling of* x*.*

Proof: Follows from Lemma 9 and definition of a suitable RMIS of G_x^+. □

4.6. The actual algorithm

Now that the $\mathcal{ABC}$-tree decomposition and the labeling rules are described, the main algorithm is relatively easy to present. We start by giving an informal description of the algorithm, along with the corresponding code, assuming that the $\mathcal{ABC}$-tree decomposition is given. Then, we give an example of execution in which the graph of Figure 5 and 6 is entirely labeled. The time complexity of the algorithm is discussed in a subsequent section, in which we also discuss the cost of computing the $\mathcal{ABC}$-tree. Then, in Section 4.8, we discuss how to turn this decision algorithm into a constructive algorithm that returns an RMIS (if one exists).

4.6.1. Description

Given the $\mathcal{ABC}$-tree T, the algorithm starts by chosing an arbitrary biconnected component $R \in \mathcal{C}$ to be the root node of T and setting it as the current node. (As already explained, if G has no biconnected component, then it is itself a tree and any property is trivially robust.) The main component of the algorithm consists of a DFS recursion within T starting from R. For every node x in T, the actual labeling of x occurs after the last child of x has been visited by the DFS, i.e., after the labels of all the children are known. At any point of the execution, if the set of labels of a node is empty *after* executing the labeling rules, this means that the current subtree (and a fortiori G itself) does not admit a robust MIS, thus the algorithm rejects (and terminates). If the execution survives until the recursion returns at the root R, then a special call to `isSatisfiable` is performed without constraints on the attachment vertex (which does not exists at R). If this call returns true, then G admits an RMIS, otherwise it does not. The code of the main algorithm is given in Algorithm 3.

The recursion of the DFS is given by the procedure in Algorithm 4, which consists of exploring the children first, then calling the labeling procedure described in Algorithm 1 on page 14, and finally reject if none of the labels were possible for the current node.

The correctness of the whole mainly follows from the correctness of the relabeling rules. Indeed, the notion of *correct labeling* (see Definition 7) specifies that the label of a node encodes the existence of RMIS in the current subtree (parametrized by the status of the highest vertex). On the other hand, Lemmas 6 to 10 guarantee that if

Algorithm 3 Main algorithm

Input: An $\mathcal{ABC}$-tree T whose nodes are $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{P}$, rooted in some $R \in \mathcal{C}$
Output: A labeling L of T

01: **foreach** $x \in$ `children`(R) **do**
02: **LabelSubTree**(x)
03: **if isSatisfiable**$(R,$ `none`$)$ **then**
04: **accept**
05: **else**
04: **reject**

Algorithm 4 Function **LabelSubTree**(x)

Parameters: A node $x \in V_T$

01: **foreach** $c \in$ `children`(x) **do**
02: **LabelSubTree**(c)
03: $L(x) \leftarrow$ **LabelNode**(x)
04: **if** $L(x) = \emptyset$ **then**
05: **reject**

all the children of a node x are correctly labeled, then the appropriate labeling rule will produce a correct labeling of x (unless the algorithm rejects). It thus follows, by induction on the tree defined by the DFS, that if the execution survives the recursion started at Lines 01 and 02 of Algorithm 3, then all the children of the root node R are correctly labeled. Lemma 9 allows us to conclude based on the call of `isSatisfiable` with parameter `none`.

4.6.2. Example

The outcome of the labeling is shown in Figure 8, corresponding to the input graph of Figure 5 and its $\mathcal{ABC}$-tree shown in Figure 6.

4.7. Time complexity

In this paragraph, we analyse the running time of the algorithm. In the following, n denotes the number of vertices in G, which is related to the number of vertices in T by a constant factor (thus the same O notation is used for both interchangeably). The purpose is not to present a fine-grained analysis. Instead, we favor simple arguments over optimal ones, the resulting running time being in any case dominated by $O(n^3)$ time steps.

Lemma 11. *The $\mathcal{ABC}$-tree decomposition can be computed in $O(n^2)$ time steps.*

Proof: The biconnected components of G can be computed using a classical algorithm by Hopcroft and Tarjan [17], which has the same complexity as a DFS in G, thus in time $O(n^2)$ in the worst case that G is dense. After this process, the components of size 1 and degree 1 correspond to the nodes in $\mathcal{P}$, the edges between components correspond to bridges is $\mathcal{B}$, and the endpoints of these bridges correspond to articulation points in $\mathcal{A}$. All of these can be determined either through looping over the components themselves (at most $O(n)$), or looping over the edges of G (at most $O(n^2)$) in order to compare the membership of their endpoints to the components (this latter test takes constant time after Hopcroft and Tarjan's algorithm). Once the four types of nodes are identified, the relations between nodes in $\mathcal{A}$ and nodes in $\mathcal{C}$ can be found by looping over all the vertices of the components ($O(n)$ vertices overall, with constant time test

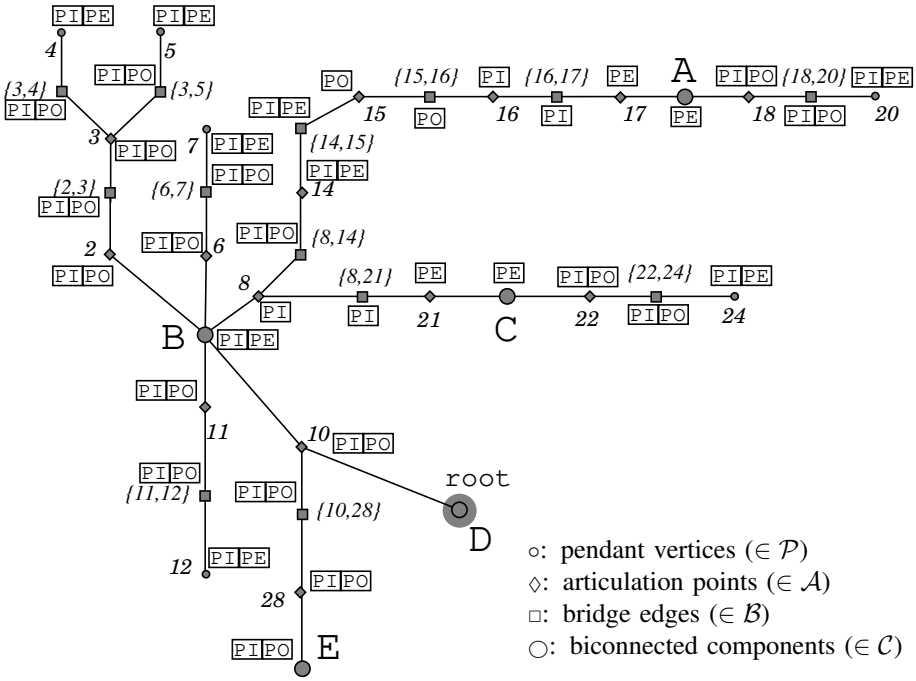

Figure 8: Outcome of the algorithm corresponding to the input graph of Figure 5.

for each) and the relations between nodes in $\mathcal{B}$ on the one hand, and nodes in $\mathcal{P}$ or $\mathcal{A}$ in the other hand, can be found by looping over all the edges of G ($O(n^2)$ edges overall, with constant time tests for each). □

Lemma 12. *Procedure* `isSatisfiable()` *takes* $O(n^2)$ *time steps.*

Proof: Building the set E^{PO} can be done by looping over all the edges of G_x and looking at the labels of its endpoints (in constant time), thus in $O(n^2)$ time steps. Building X without these edges is also linear in the number of edges, thus costs $O(n^2)$ time steps. Then, testing if a graph is bipartite can be done by performing a 2-coloring of it along a BFS as follows: give color 1 to an arbitrary first vertex v, then as the BFS progresses, give color 2 to all the vertices at distance 1 from v, then color 1 again to all the vertices at distance 2 from v, etc. Possibly restart the BFS algorithm if X is not connected. As the BFS algorithm runs in time linear in the number of edges (whether or not X is connected), the 2-coloring algorithm will take up to $O(n^2)$ time steps, and the detection of whether the coloring is proper can also be done in $O(n^2)$ time steps (checking the two endpoints of every edge). If the procedure is still running (X is bipartite), the obtained 2-coloring can be reused directly over lines 07 and 08 to assign to every vertex its SAT constraint x_i or $\neg x_i$, depending on its color and connected component (the latter being possibly determined during the BFS). Then, Lines 09 to 13 consist of a loop over (part of) the vertices with constant time processing time for each. Lines 14 and 15 consist of a loop over the edges of E^{PO} and constant processing time for each, thus at most $O(n^2)$. Lines 16 to 19 take constant time. Finally, the resulting formula is made of $O(n^2)$ constraints over $O(n)$ clauses, and it is known that 2-SAT is solvable in time linear in the number of constraints. In conclusion, the overall procedure takes $O(n^2)$ time steps. □

Lemma 13. *The decision algorithm presented in Section 4.2 takes $O(n^3)$ time steps.*

Proof: The main loop of the algorithm performs a depth first search in the $\mathcal{ABC}$-tree T, whose running time is $O(n)$. Every node x is labeled once, after the DFS has finished the exploration of its subtree. At that moment, four cases apply depending on whether x is of type $\mathcal{A}, \mathcal{B}, \mathcal{C}$, or $\mathcal{P}$. If x is of type $\mathcal{P}$, the labeling takes constant time (see Algorithm 1 Lines 03 to 04, and 17 to 25, respectively). If x is of type $\mathcal{B}$, the labeling also takes constant time because x has only one child (Lines 17 to 25). If x is of type $\mathcal{A}$, then the labeling rule performs two loops over the children of x, thus the labeling takes $O(n)$ time steps (the nested existential quantifier can be tested upon the same loop). If x is in $\mathcal{C}$, then two or three calls are made to `isSatisfiable()`, whose running time is $O(n^2)$ by Lemma 12. As a result, the overall running time is dominated by the latter being applied to possibly $O(n)$ nodes, for a total of $O(n^3)$ time steps. □

4.8. Constructivity

The algorithm presented over the previous subsections is a decision algorithm that returns yes or no, or equivalently, accepts or rejects the given instance. However, the algorithm is fairly easy to adapt in order to obtain an actual RMIS if one exists. There are essentially two options: either the RMIS is constructed in parallel of the decision algorithm, or a dedicated algorithm traverses again the tree after the algorithm has completed. Here, we briefly sketch both solutions.

1. *Construction in parallel.* In this version, the main difficulty is that the algorithm cannot decide which of the current labels will eventually be compatible with later (higher) constraints in the tree. For example, a node x may admit an RMIS in its subtree whether or not its attachment vertex $v(x)$ is included (label *PI* and *PO*), but it might later become mandatory that $v(x)$ is included. Thus, the idea is to build several RMIS simultaneously, one for each possible label of the current attachment vertex. These RMIS can be built by extending the ones of the children (with compatible status), starting from the leaves, thus at most two RMIS need to be memorized for the current subtree. In the particular case that x is a biconnected component, it is needed that the satisfying assignment found by the 2-SAT solver be known and transposed into a membership status for each vertex (similarly to the set S in the proof of Lemma 9).

2. *Construction afterwards.* In this version, we are given an already labeled tree whose root component has been solved (successfully). The algorithm consists of traversing again the labeled tree, by making *arbitrary choices* whenever the considered attachment vertex has several labels. Indeed, the existence of an RMIS corresponding to each such label is guaranteed by the labelling. The cases that x is of type $\mathcal{A}, \mathcal{B}$, or $\mathcal{P}$ are trivial (just pick one of the labels of the child and translate it as a membership status for its attachment vertex). The only difficulty is when x is a biconnected components. In this case, a label is chosen arbitrarily among the labels of its attachment vertex, and the 2-SAT reduction procedure is called again with the configuration corresponding to this constraint in order to obtain the satisfying assignement that specifies which inner vertex must be included to the MIS. (Alternatively, the satisfying assignments might be recorded while in the decision algorithm.)

A possible outcome of the constructive algorithm, obtained by manually following the first method is shown in Figure 9.

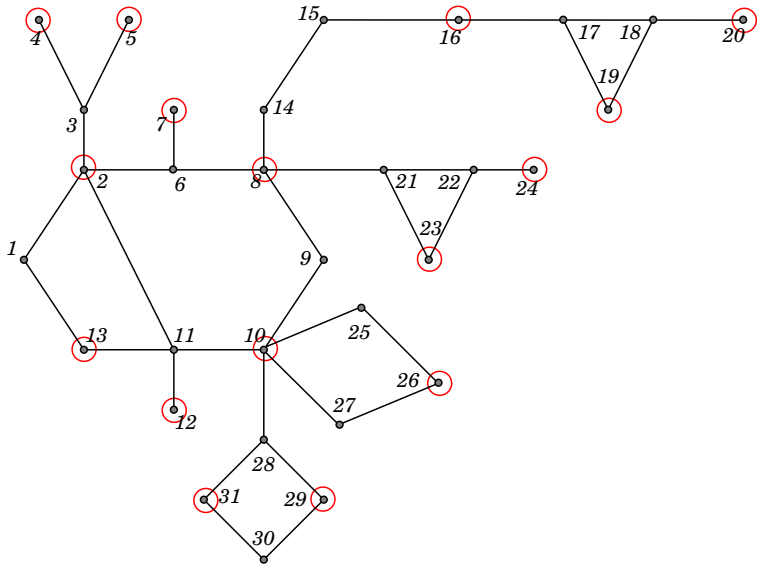

Figure 9: A possible RMIS of G that includes 16 vertices.

5. Further discussion on the temporal interpretation of robustness

As discussed in the introduction, the original motivation behind the concept of robustness comes from highly-dynamic networks. Here, we review some of the background that led to its definition. As the content pertains to the temporal interpretation, the reader interested in the notion of robustness *per se* can safely omit reading this section. In [9], three canonical ways of redefining combinatorial problems in highly-dynamic networks were explored, in particular covering problems such as *dominating set*, *vertex cover*, or *independent set*. The main focus in [9] was on dominating sets, which are subsets of nodes $S \subseteq V$ such that each node in the network is either in S, or has a neighbor in S. Three natural adaptations of these problems were formulated:

- *Temporal* version: the covering property of the problem is achieved *over time* – *E.g.,* for domination, every node outside the set must share an edge *at least once* over the lifetime with a node in the set.
- *Evolving* version: the covering property must be satisfied at any given instant, relative to the current structural snapshot of the network; however, the solution can be updated as the structure of the network changes. This version is closer to the traditional "dynamic graph algorithms" (also referred to as "reoptimization").
- *Permanent* version: the covering property must be satisfied in every snapshot (as for the evolving version), but here the solution cannot be updated.

The three versions are related. For example, in the case of dominating sets, the temporal version consists of computing a dominating set in the *footprint* of the network. The footprint, sometime denoted $\cup\mathcal{G}$, is the union of all snapshots, i.e., it contains every edge that is present in at least one snapshot. By contrast, the permanent version consists of computing a solution in $\cap\mathcal{G}$ (the intersection of the snapshot sequence) [9]. Furthermore, solutions to the *permanent* and the *temporal* versions form upper and lower bounds for the *evolving* version, respectively.

In [11], the temporal definition above is extended to *infinite* lifetime networks, by requiring that the covering relation (domination or else) be satisfied *infinitely often*. The authors observe that whenever the network is temporally connected in a recurrent way, which corresponds to Class $\mathcal{TC}^{\mathcal{R}}$ in [7] (and Class 5 in [8]), one has the guarantee that among all the edges that appear at some point, at least a connected spanning subset of them must reappear forever [5]. In other words, an equivalent characterization of $\mathcal{TC}^{\mathcal{R}}$ is that the *eventual footprint* of the network (i.e., the union of those edges which always reappear) is a connected spanning subgraph of the footprint.

In summary, if the network is assumed to be in $\mathcal{TC}^{\mathcal{R}}$, then one has the guarantee that at least a connected spanning subset of the edges are recurrent, but this subset is not known (even if the footprint is known). The very concept of robustness is to overcome this uncertainty by ensuring that the computed solution is valid for every possible such subsets of a given footprint, *i.e.* every possible eventual footprint, provided that the network is in $\mathcal{TC}^{\mathcal{R}}$.

For completeness, let us cite a few recent works that considered the problem of computing temporal covering structures in highly-dynamic networks (or graphs), although not related to robustness. Mandal et al. [21] study approximation algorithms for the permanent version of *dominating sets*. Bamberger *et al.* [2] also consider a temporal variant of vertex coloring and MIS. Finally, Akrida *et al.* [1] define a variant of the temporal version in the case of vertex cover, in which a solution is not just a set of nodes (as it was in [9]) but a set of pairs $(nodes, times)$, allowing different nodes to cover the edges at different times (and within a sliding time window).

6. Concluding remarks

This paper introduced a new form of heredity called robustness, motivated by various kinds of dynamic networks. In particular, we believe that robustness is a key property of highly dynamic systems for achieving stable structures in unstable environments.

Focusing on the classical covering problem MIS, we characterized the set of graphs in which all MISs are robust. We gave partial characterizations of the existential analogues of this class, namely graphs that *admit* a robust solution. We characterized the class entirely by means of a polynomial time algorithm which finds a robust MIS in an arbitrary graph is one exists (and rejects otherwise). Whether a characterization of the existential classes exists in terms of elementary graph properties is an open question. It would also be interesting to investigate the robustness of other types of structures (e.g. minimal dominating sets) and of basic graph properties, with the aim to understand robustness at a deeper level. For example, bipartiteness or "sputnikness" are themselves robust properties of a graph.

Acknowledgment

This research has been supported by ANR project ESTATE (ANR-16-CE25-0009-03). We thank the anonymous referees for their many helpful comments on an earlier version of this article.

References

[1] Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Viktor Zamaraev. Temporal vertex cover with a sliding time window. *CoRR*, abs/1802.07103, 2018.

[2] Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Local distributed algorithms in highly dynamic networks. *arXiv preprint arXiv:1802.10199*, 2018.

[3] Béla Bollobás and Andrew Thomason. Hereditary and monotone properties of graphs. In *The Mathematics of Paul Erdös II*, pages 70–78. Springer, 1997.

[4] Mieczysław Borowiecki, Izak Broere, Marietjie Frick, Peter Mihok, and Gabriel Semanišin. A survey of hereditary properties of graphs. *Discussiones Mathematicae Graph Theory*, 17(1):5–50, 1997.

[5] Nicolas Braud-Santoni, Swan Dubois, Mohamed-Hamza Kaaouachi, and Franck Petit. The next 700 impossibility results in time-varying graphs. *International Journal of Networking and Computing*, 6(1):27–41, 2016.

[6] Sonja Buchegger and J-Y Le Boudec. Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks. In *Proc. of 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 403–410. IEEE, 2002.

[7] Arnaud Casteigts. Finding structure in dynamic networks. *CoRR*, abs/1807.07801, 75p, 2018.

[8] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012.

[9] Arnaud Casteigts, Bernard Mans, and Luke Mathieson. On the feasibility of maintenance algorithms in dynamic graphs. *CoRR*, abs/1107.2722, 2011.

[10] Reuven Cohen and Shlomo Havlin. *Complex networks: structure, robustness and function*. Cambridge university press, 2010.

[11] Swan Dubois, Mohamed-Hamza Kaaouachi, and Franck Petit. Enabling minimal dominating set in highly dynamic distributed systems. In *17th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 51–66, 2015.

[12] Afonso Ferreira. Building a reference combinatorial model for manets. *IEEE network*, 18(5):24–29, 2004.

[13] Zoltán Füredi. The number of maximal independent sets in connected graphs. *Journal of Graph Theory*, 11(4):463–470, 1987.

[14] Jianxi Gao, Sergey V Buldyrev, Shlomo Havlin, and H Eugene Stanley. Robustness of a network of networks. *Physical Review Letters*, 107(19):195701, 2011.

[15] Jerrold R Griggs, Charles M Grinstead, and David R Guichard. The number of maximal independent sets in a connected graph. *Discrete Mathematics*, 68(2-3):211–220, 1988.

[16] F Harary. *Graph theory*. Addison-Wesley Reading MA USA, 1969.

[17] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[18] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

[19] Valerie King. Lower bounds on the complexity of graph properties. In *Proc. of the twentieth annual ACM symposium on Theory of computing (STOC)*, pages 468–476. ACM, 1988.

[20] Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *CoRR*, abs/1710.04073, 2017.

[21] Subhrangsu Mandal and Arobinda Gupta. Approximation algorithms for permanent dominating set problem on dynamic networks. In *International Conference on Distributed Computing and Internet Technology*, pages 265–279. Springer, 2018.

[22] John W Moon and Leo Moser. On cliques in graphs. *Israel journal of Mathematics*, 3(1):23–28, 1965.