A certifying and dynamic algorithm for the recognition of proper circular-arc graphs

Francisco J. Soulignac [*]

CONICET and Departamento de Ciencia y Tecnología, Universidad Nacional de Quilmes, Buenos Aires, Argentina.

Abstract

We present a dynamic algorithm for the recognition of proper circular-arc (PCA) graphs, that supports the insertion and removal of vertices (together with its incident edges). The main feature of the algorithm is that it outputs a minimally non-PCA induced subgraph when the insertion of a vertex fails. Each operation cost $O(\log n + d)$ time, where n is the number vertices and d is the degree of the modified vertex. When removals are disallowed, each insertion is processed in $O(d)$ time. The algorithm also provides two constant-time operations to query if the dynamic graph is proper Helly (PHCA) or proper interval (PIG). When the dynamic graph is not PHCA (resp. PIG), a minimally non-PHCA (resp. non-PIG) induced subgraph is obtained.

Keywords: dynamic representation, certifying algorithm, proper circular-arc graphs, proper interval graph, proper Helly circular-arc graphs.

1 Introduction

A *circular-arc (CA) model* is a family of arcs of a circle. A graph G *admits* a CA model $\mathcal{M}$ when its vertices are in a one-to-one correspondence with the arcs of $\mathcal{M}$ in such a way that two vertices of G are adjacent if and only if their corresponding arcs have a nonempty intersection. Those graphs that admit a CA model are called *circular-arc (CA) graphs.* Proper circular-arc graphs and proper interval graphs form two of the most studied subclasses of CA graphs. A CA model $\mathcal{M}$ is *proper* when no arc of $\mathcal{M}$ is properly contained in another arc of $\mathcal{M}$, while $\mathcal{M}$ is an *interval (IG) model* when the union of its arcs does not cover the entire circle. A graph is a *proper circular-arc (PCA) graph* when it admits a proper CA model, while it is a *proper interval (PIG) graph* when it admits a proper IG model.

The *(static) recognition problem* for PCA (resp. PIG) graphs asks if an input graph G is PCA (resp. PIG). A recognition algorithm that outputs YES or NO is not that useful in practice for two reasons. First, there are many applications in which PIG and PCA models of G are looked for, while several algorithms work more efficiently when a PIG or PCA model of G is available [10, 15, 28]. Second, and not less important, a buggy implementation can lead to incorrect answers that a user cannot corroborate. A *certifying* recognition algorithm yields a *witness* W proving that the output is correct for G. Besides proving correctness, two additional properties are required for W [25]. First, there exists a *checker* with a “trivial” implementation that, given G and W, *authenticates* that the output is correct for G. Second, there is a simple proof that the existence of W implies the output on G. With these two ingredients, a user can test that the output for G is correct, even when the implemented algorithm has bugs [25].

[*]mail: `francisco.soulignac@unq.edu.ar`

Witnesses are classified as *positive* or *negative* according to the output given for G. The former prove that G is PCA (YES output), while the latter prove that G is not PCA (NO output). A priori, there are many certifying algorithms for the recognition of PCA graphs, as we can chose different kinds of witnesses. Although all of them can be used to authenticate the output, they need not be equally useful for the user. This statement is obvious in those application where the goal is to produce a PCA model of G, but it also holds for those applications on PCA graphs that require a specific kind of input. Thus, a positive witness with the required interface is better than one that has to be further processed. Similarly, a negative witness should highlight the reason why G is not PCA. Arguably, PCA models are the most useful positive witnesses, while *minimally forbidden subgraphs* are the most useful negative witnesses.

Unfortunately, the positive witnesses that we use in this article are not PCA models, but *round representations* [8]. Roughly speaking, a round (resp. straight) representation Φ is like a PCA (resp. PIG) model in which the actual position of the arcs is missing. Instead, we know the order of the arcs and which are the leftmost and rightmost arcs intersected by a given arc. Fortunately, Φ is enough for all those applications in which knowing the actual position of an arc is not required, e.g. [15]. Also, it is trivial to obtain a PCA (PIG) model $\mathcal{M}$ *associated* to Φ in $O(n)$ time, where n is the number of arcs of $\mathcal{M}$; by *associated*, we mean that the arcs of Φ and $\mathcal{M}$ appear in the same order.

In this article we consider a *dynamic* version of the recognition problem for PCA graphs. The goal is to keep a round representation Φ of a graph G while some operations are applied. We allow two kinds of updates: the insertion of a new vertex (and the edges incident to it), and the removal of an existing vertex (and its incident edges). Those insertions that yield non-PCA graphs have no effects on Φ; instead, an error message is obtained. Also, the algorithm must answer if G is PIG or not and, if affirmative, then Φ must be a straight representation. Consequently, Φ can be immediately applied on algorithms that work on PIG graphs. When efficiency does not matter, the dynamic problem is solved by applying any static recognition algorithm for each update. The idea, however, is to reduce the complexity of the operations.

To motivate the development of dynamic algorithms for PIG graphs, Hell et al. [13] describe an application to *physical mapping* of DNA. The problem is to find a straight representation Φ of an input graph G that encodes some biological data, or to prove that no such model exists. As time goes by, further experiments may prove that the initial biological data is not accurate. The resulting changes in the data correspond to the insertion and removal of vertices and edges from G. Instead of building a new straight representation from scratch, the goal is to "fix" Φ efficiently.

The concerns about the reliability and usefulness of the outputs, that we had for the static recognition algorithms, hold also for the dynamic ones. The existence of a round representation Φ proves that G is a PCA graph, thus Φ can be taken as the positive witness. However, when the algorithm rejects an update claiming that it leads to a non-PCA graph, can we trust this claim blindly? And, even if we do trust, we still want a negative witness to check if the input data is incorrect. This is particularly true for the above application to physical mapping of DNA, since we expect the experiments to be inaccurate at some point, and we cannot assume the erroneous data yields a PCA graph. A *certifying* and dynamic algorithm for the recognition of PCA graphs outputs a minimally forbidden subgraph when some update is rejected.

Authenticating that a round representation Φ encodes G or that F is minimally forbidden subgraph of G are trivial tasks, as desired. However, the time required for these authentications is linear on the size of G. Thus, we cannot expect the user to authenticate the witnesses after each operation, as doing so throws out the efficiency benefits of the dynamic algorithm. The difference between static and dynamic algorithms is that the latter are not, strictly speaking, algorithms. Instead, they are *abstract data types* that keep a certain *data structure* that reacts to different operations. Thus, Φ is not given as output when an insertion or removal is applied and, so, Φ should be authenticated against G only occasionally.

We can conceive three types of checkers, which we call *static*, *dynamic*, and *monitors*. Static checkers are static algorithms that authenticate the witness against the static graph G. Dynamic checkers are also static algorithms, but they check one update of the dynamic algorithm against the round representation Φ. Finally, monitors are dynamic algorithms that ensure the correctness of data structure $\tilde{\Phi}$ implementing Φ [2, 25]. Thus, a monitor is an abstract data type that sits between the user and the recognition algorithm. The user interacts with the monitor as if it were a round representation. In turn, the monitor forwards each operation to $\tilde{\Phi}$, while it checks the correct behavior of $\tilde{\Phi}$ and the generated output. In case of an error, the monitor raises an exception. The main difference between checkers and monitors is that the latter may require access to operations that the are restricted to the user. Checkers are usually simpler, as they have no knowledge of $\tilde{\Phi}$, and can be implemented even when the source code of the recognition algorithm is unavailable. However, the same reason could make them less efficient. Thus, checkers and monitors are complementary tools.

Previous work. Linear-time algorithms for generating PIG models of graphs are known since more than twenty years, e.g. [4, 8, 14]. While dealing with the correctness of their algorithm, Deng et al. [8] prove that a minimally forbidden subgraph F must exist when the algorithm fails. Although it is not discussed in [8], F can be obtained in $O(n)$ time. A second way to find a F is to apply the dynamic recognition algorithm by Hell et al. [13]. In a first phase, the algorithm finds a set of vertices V such that the subgraph $G[V]$ induced by V is PIG, and $G[V \cup \{v\}]$ is not PIG. In a second phase, the algorithm transforms $G[V \cup \{v\}]$ into F by removing vertices from V. This strategy, which is discussed in [30] for PCA graphs, costs linear time for PIG graphs. A similar approach using only incremental graphs is discussed in [25] for planar graphs and in [21] for interval graphs. Arguably, the simpler linear time algorithm to find F was presented in 2004 by Hell and Huang [12], who extend the LexBFS algorithm by Corneil [3] to exhibit such a forbidden when the input is not PIG. Meister [26] also applies LexBFS to find a negative witness, but this witness is not always a minimally forbidden subgraph.

The problem of building a PCA model of a graph G is also well settled, but fewer algorithms are known [8, 20, 30]. Regarding the certification problem, Hell and Huang [12] show how to obtain a minimally forbidden subgraph F when G is co-bipartite and not PCA. The first algorithm that shows how to obtain a negative witness when G is not co-bipartite was presented in 2009 by Kaplan and Nussbaum [20]. Unfortunately, their witnesses are not forbidden subgraphs, but odd cycles of incompatibility graphs. Up to this date, the only algorithm that is able to compute F in linear time for every PCA graphs was given by the author [30] in 2015. The idea is to apply a dynamic recognition algorithm in two phases as discussed above.

Lin and Szwarcfiter [23] survey different algorithms for the recognition of other classes of circular-arc graphs, while McConnell et al. [25] discuss a theoretical framework for certifying algorithms and explain why they are preferred over non-certifying ones. McConnell et al. survey certifying recognition algorithms for other classes of graphs as well.

In the last years, dynamic recognition algorithms for many classes of graphs were developed [5–7, 9, 11, 17–19, 27, 29, 31]. Among these examples, the only one providing negative witnesses is the one by Crespelle and Paul [6] for the recognition of directed cographs. We remark that the minimally forbidden subgraphs for directed cographs have $O(1)$ vertices, thus they are generated when required. On the other hand, a minimally forbidden subgraph for PCA graphs can have $\Theta(n)$ vertices with degree $O(1)$. Thus, the computation of such a forbidden is dynamic.

Type of data structure	ep^+	ep^-
Incremental	$O(1)$	$O(n)$
Decremental	$O(n)$	$O(1)$
Fully dynamic	$O(\log n)$	$O(\log n)$

Table 1: The actual values of ep^+ and ep^- according to the data structure employed

Our results. We conceive our manuscript as the forth in a series of articles. The series begins in 1996 with the recognition algorithm for PCA graphs developed by Deng et al. [8]. As part of their algorithm, Deng et al. devise a vertex-only incremental algorithm for the recognition of connected PIG graphs that runs in $O(d)$ time per vertex insertion, where d is the degree of the inserted vertex. The data structure that supports their recognition algorithm is a straight representation. The second article of the series dates back to 2001, where Hell et al. [13] extend the algorithm by Deng et al. to solve the dynamic recognition of PIG graphs. Their algorithm runs in $O(d+\mathrm{ep}^+)$ time per vertex insertion, $O(d+\mathrm{ep}^-)$ time per vertex removal, $O(\mathrm{ep}^+)$ time per edge insertion, and $O(\mathrm{ep}^-)$ time per edge removal. The values of ep^+ and ep^- depend on the data structure employed to implement the straight representations, as depicted in Table 1. Note that the algorithm is optimal if only insertions are allowed, while it is almost optimal when both operations are allowed. Indeed, Hell et al. prove that at least $\Omega(\log n/(\log\log n + \log b))$ amortized time is required by the fully dynamic algorithm in the cell probe model of computation with word size b. Finally, in 2015, the author [30] extended the algorithm by Hell el al. for the recognition of PCA graphs. The algorithm works with round representations and has the same complexity as the one by Hell et al. Moreover, the round representation is straight when the input graph is PIG, thus the algorithm solves the dynamic recognition of PIG graphs as well.

In this article we further extend the algorithm in [30] to provide a certifying and dynamic algorithm for the recognition of PCA graphs as discussed above. The algorithm is restricted to the insertion and removal of vertices, and we ignore the problem for edge operations. Specifically, the algorithm implements a round representation Φ of the input graph G, and it yields a minimally forbidden subgraph when a vertex insertion fails. Our algorithm is as efficient as the one by Hell et al., as it handles the insertion of a new vertex v in $O(d+\mathrm{ep}^+)$ time, while the removal of v costs $O(d+\mathrm{ep}^-)$ time. The user can also ask, at any point, if G is a PIG graph; this query costs $O(1)$ time. If affirmative, then Φ is a straight representation of G. Otherwise, a minimally forbidden subgraph is obtained.

We remark that when the insertion of a vertex v with $O(1)$ neighbors fails, the minimally forbidden subgraph F can be of size $\Theta(n)$. However, only $O(\mathrm{ep}^+)$ time is available to generate F. Thus, besides keeping Φ, the dynamic algorithm stores a *partial* forbidden subgraph $\mathcal{P}(\Phi)$. When an insertion fails, $\mathcal{P}(\Phi)$ is extended with v to yield F. This scheme is similar to the one used for the positive witness. The difference is that $\mathcal{P}(\Phi)$ is not accessible by the user, who observes the dynamic algorithm as an implementation of Φ. As is the case with Φ, the output F must provide an efficient and convenient interface to the user. Of course, because of the inevitable aliasing between F and G, no updates on F are possible, and any modification on G invalidates F. If required, a copy of F can be obtained in $O(|E(F)|) = O(n)$ time.

Organization of the manuscript. Section 2 introduces the basic terminology and notation. Section 3 presents the *Reception Theorem*, which characterizes when a graph H is PCA knowing that $H \setminus \{v\}$ is PCA. The Reception Theorem sums up the results we require from [8, 13, 30], and guides the certifying algorithm that we develop later. Section 4 describes the data structure that we employ, and the algorithms required to update the partial forbidden subgraph $\mathcal{P}(\Phi)$. Section 5 shows the certifying algorithm we use when a vertex is inserted. Section 6 depicts the complete recognition algorithm, including the insertion and removal of vertices and the query for the recognition of PIG graphs. Section 6 also discusses the authentication problems. Finally, Section 7 has some further remarks and open problems.

2 Preliminaries

For a graph G, we use $V(G)$ and $E(G)$ to denote the sets of vertices and edges of G, respectively, and $n = |V(G)|$ and $m = |E(G)|$. The *neighborhood* of a vertex v is the set $N_G(v)$ of all the neighbors of v, while the *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. If $N_G[v] = V(G)$, then v is a *universal vertex*, while if $N_G(v) = \emptyset$, then v is an *isolated vertex*. Two vertices v and w are *twins* when $N_G[v] = N_G[w]$. The cardinality of $N_G(v)$ is the *degree* of v and is denoted by $d_G(v)$. We omit the subscripts from N and d when there is no ambiguity about G.

The subgraph of G *induced* by $V \subseteq V(G)$ is the graph $G[V]$ that has V as its vertex set, where two vertices of $G[V]$ are adjacent if and only if they are adjacent in G. A *clique* is a subset of pairwise adjacent vertices. We also use the term *clique* to refer to the corresponding subgraph. An *independent set* is a set of pairwise non-adjacent vertices. A *semiblock* of G is a nonempty set of twin vertices, and a *block* of G is a maximal semiblock.

The *complement* of G, denoted by $\overline{G}$, is the graph that has the same vertices as G and such that two vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G. Each component of $\overline{G}$ is called a *co-component* of G, and G is *co-connected* when $\overline{G}$ is connected. The *union* of two vertex-disjoint graphs G and H is the graph $G \cup H$ with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. The *join* of G and H is the graph $G + H = \overline{\overline{G} \cup \overline{H}}$, i.e., $G + H$ is obtained from $G \cup H$ by inserting all the edges vw, for $v \in V(G)$ and $w \in V(H)$.

A graph G is *bipartite* when there is a partition V_1, V_2 of $V(G)$ such that both V_1 and V_2 are independent sets. Contrary to the usual definition of a partition, we allow one of the sets V_1 and V_2 to be empty. So, the graph with one vertex is bipartite for us. The partition of $V(G)$ into V_1, V_2, denoted by $\langle V_1, V_2 \rangle$, is called a *bipartition* of G. When $\overline{G}$ is bipartite, G is a *co-bipartite* graph and each bipartition of $\overline{G}$ is a *co-bipartition* of G.

For $v \in V(G)$ and $W \subseteq V(G)$, we say that v and W are: *adjacent* if $N(v) \cap W \neq \emptyset$, *co-adjacent* if $W \setminus N(v) \neq \emptyset$, and *fully adjacent* if $W \subseteq N(v)$. Two disjoint semiblocks B and W are *adjacent* if some vertex in B is adjacent to some vertex in W; observe that B and W are adjacent if and only if every vertex in B is fully adjacent to W. If $B \cup W$ is a semiblock, then B is a *twin* of W. A semiblock B is *universal* when its vertices are universal, and *isolated* when the vertices in B are not adjacent to $V(G) \setminus B$. For a disjoint family of semiblocks $\mathcal{B}$, the subgraph $G[\mathcal{B}]$ of G *induced by* $\mathcal{B}$ is obtained from $G[\bigcup \mathcal{B}]$ by removing all but one vertex from each semiblock of $\mathcal{B}$. Clearly, $G[\mathcal{B}]$ is an induced subgraph of G.

2.1 Orderings and ranges

In this article, an *order* is a pair (S, R) where S is a finite, and possibly empty, set that admits an enumeration $X = x_1, \ldots, x_n$ of its elements such that $R(x_i) = x_{i+1}$ for every $1 \leq i < n$ and $R(x_n) \in \{x_1, \perp\}$ for some undefined value $\perp \notin S$. We say that (S, R) is *linear* when $R(x_n) = \perp$, while (S, R) is *circular* when $R(x_n) = x_1$. When (S, R) is linear, x_1 and x_n are the *leftmost* and *rightmost* elements of (S, R). The enumeration X of S is said to be an *ordering* of (S, R).

Clearly, every enumeration X of a finite set S defines a linear order and a circular order (S, R), both of which have X as its ordering. Thus, we say that X is a *linear (resp. circular) ordering* to mean that (S, R) is a linear (resp. circular) order. In such a case, we write R^X as a shortcut for R; we omit the superscript X when X is clear by context. For each $x \in X$, the element $R(x)$ is the *right near neighbor* of x. When we want to make no distinctions about

X being linear or circular, we simply state that X is an *ordering*. Note, however, that every ordering is either linear or circular, and it cannot be both at the same time.

Every ordering $X = x_1, \ldots, x_n$ that we shall consider is embedded in some larger circular order. Hence, all the operations on the indices of X are taken modulo n, regardless of whether X is linear or circular. We use the standard interval notation applied to orders, though we call them *ranges* to avoid confusions with interval graphs. So, the *range* $[x_i, x_j]$ of X is the linear ordering $x_i, \ldots, x_j$ where, as stated before, $x_1 = x_{n+1}$. Similarly, the *range* $[x_i, x_j)$ is obtained by removing the rightmost element of $[x_i, x_j]$, the *range* $(x_i, x_j]$ is obtained by removing the leftmost element from $[x_i, x_j]$, and the *range* (x_i, x_j) is obtained by removing both the leftmost and rightmost elements from $[x_i, x_j]$. The *reverse* of X is the ordering $X^{-1} = x_n, \ldots, x_1$, where X^{-1} is linear if and only if X is linear. We write L^X as a shortcut for $R^{X^{-1}}$ and we omit X when it is clear from context. Note that $L(x_{i+1}) = x_i$ for every $1 \leq i < n$, while $L(x_1) \in \{x_n, \bot\}$ equals x_n if and only if X is circular. For each $x \in X$, the element $L(x)$ is the *near left neighbor* of x. If X and $Y = y_1, \ldots, y_m$ are linear orderings, then $X \bullet Y$ denotes the linear ordering $x_1, \ldots, x_n, y_1, \ldots, y_m$.

The range notation that we use for ranges clashes with the usual notation for ordered pairs. Thus, we write $\langle x, y\rangle$ to denote the ordered pair (x, y). The unordered pair formed by x and y is, as usual, denoted by $\{x, y\}$. Also, for the sake of notation, we sometimes write $\#S$ to denote the cardinality of a range S. For any function f, we write f^0 to mean the identity and $f^{k+1}(x) = f^k(f(x))$.

2.2 Contigs, round representations, and proper circular-arc graphs

In this section we present an alternative definition of proper circular-arc graphs that follows from [8, 16]. These definitions are based on the notion of round representations, which are combinatorial views of proper circular-arc models; see [1].

A *contig* is a pair $\phi = \langle \mathcal{B}(\phi), F_r^\phi\rangle$ where $\mathcal{B}(\phi) = B_1, \ldots, B_n$ is an ordering of pairwise disjoint sets, and F_r^ϕ is a mapping from $\mathcal{B}(\phi)$ to $\mathcal{B}(\phi)$ such that:

(i) $F_r^\phi(B_i) \in (B_i, F_r^\phi(B_{i+1})]$, for every $1 \leq i < n$,

(ii) if $\mathcal{B}(\phi)$ is linear, then $F_r^\phi(B_n) = B_n$; otherwise $F_r^\phi(B_n) \in [B_1, F_r^\phi(B_1)]$, and

(iii) $B_i \notin (F_r^\phi(B_i), F_r^\phi(F_r^\phi(B_i))]$ for every $1 \leq i \leq n$.

We classify each contig ϕ as either linear or circular according to whether $\mathcal{B}(\phi)$ is linear or circular. Note that ϕ is linear if and only if $F_r^\phi(B_n) = B_n$.

We use a convenient notation for dealing with the range $(B, F_r^\Phi(B)]$. For $B, W \in \mathcal{B}(\phi)$, we write $B \longrightarrow_\phi W$ to mean that $W \in (B, F_r^\phi(B)]$. Similarly, we write $B \not\longrightarrow_\phi W$ to indicate that $W \notin (B, F_r^\phi(B)]$, and $B \overset{=}{\longrightarrow}_\phi W$ to indicate that either $B = W$ or $B \longrightarrow_\phi W$. With the $\longrightarrow$-notation, we can rewrite conditions (i)–(iii) as follows:

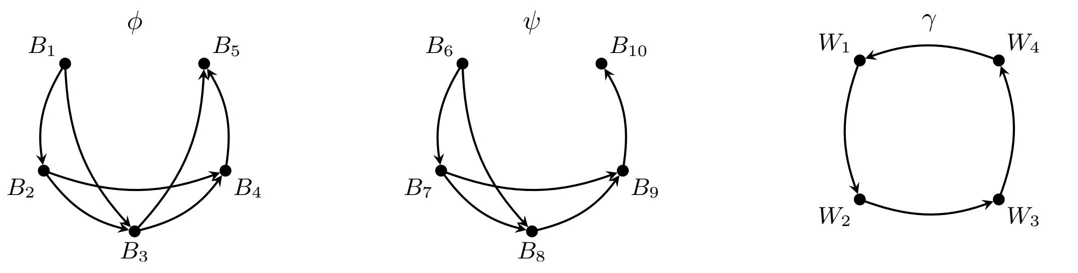
(i) $B_i \longrightarrow_\phi B_{i+1}$ and $B_{i+1} \overset{=}{\longrightarrow}_\phi F_r(B_i)$ for every $1 \leq i < n$,

(ii) if ϕ is linear, then $B_n \not\longrightarrow_\phi B_1$; otherwise, $B_n \longrightarrow_\phi B_1$ and $B_1 \overset{=}{\longrightarrow}_\phi F_r(B_n)$, and

(iii) either $B_i \not\longrightarrow_\phi B_j$ or $B_j \not\longrightarrow_\phi B_i$ for every $1 \leq i \leq j \leq n$.

Figure 1 depicts three contigs with their corresponding $\longrightarrow$ relation.

The sets in $\mathcal{B}(\phi)$ are referred to as *semiblocks* of ϕ, while $V(\phi) = \bigcup \mathcal{B}(\phi)$ is the set of *vertices* of ϕ. For simplicity, we write L^ϕ and R^ϕ as shortcuts for $L^{\mathcal{B}(\phi)}$ and $R^{\mathcal{B}(\phi)}$. Recall that $L^\phi(B)$ and $R^\phi(B)$ are the left and right near neighbors of B for every $B \in \mathcal{B}(\phi)$. Similarly, we say



$\mathcal{B}(\phi) = B_1, B_2, B_3, B_4, B_5$	$\mathcal{B}(\psi) = B_6,\ B_7\ ,\ B_8\ ,\ B_9\ , B_{10}$	$\mathcal{B}(\gamma) = W_1, W_2, W_3, W_4$
$R^\phi \quad = B_2, B_3, B_4, B_5,\ \bot$	$R^\psi \quad = B_7,\ B_8\ ,\ B_9\ , B_{10},\ \bot$	$R^\gamma \quad = W_2, W_3, W_4, W_1$
$F_r^\phi \quad = B_3, B_4, B_5, B_5, B_5$	$F_r^\psi \quad = B_8,\ B_9\ ,\ B_9\ , B_{10}, B_{10}$	$F_r^\gamma \quad = W_2, W_3, W_4, W_1$
$U_r^\phi \quad = B_4, B_5,\ \bot\ ,\ \bot\ ,\ \bot$	$U_r^\psi \quad = B_9, B_{10}, B_{10},\ \bot\ ,\ \bot$	$U_r^\gamma \quad = W_3, W_4, W_1, W_2$

Figure 1: Two linear contigs ϕ and ψ and a circular contig ρ. Each contig $\bullet$ is depicted with their corresponding $\longrightarrow_\bullet$ relations that corresponds to the semiblock ordering $\mathcal{B}(\bullet)$ and the mappings $R^\bullet$ and $F_r^\bullet$. The mapping $U_r^\bullet = R^\bullet(F_r^\bullet)$ is also shown.

that $F_r^\phi(B)$ is the *right far neighbor* of B. The *left far neighbor* of B is the unique semiblock $F_l^\phi(B)$ such that $W \longrightarrow_\phi B$ if and only if $W \in [F_l^\phi(B), B)$, for every $W \in \mathcal{B}(\phi)$. Two other mappings are highly used in this manuscript: $U_r^\phi(B) = R^\phi(F_r^\phi(B))$ and $U_l^\phi(B) = L^\phi(F_l^\phi(B))$ are the *unreached right* and *unreached left* semiblocks, respectively. As usual, we do not write the subscript and superscript ϕ for L, R, F_l, F_r, U_l, U_r, and $\longrightarrow$ when ϕ is clear by context. Figure 1 shows the values of R, F_r, and U_r for some contigs. Note that ϕ is linear if and only if $R(B) = \bot$ (and $F_r(B) = B$) for some semiblock B.

A *round representation* is a family $\Phi = \{\phi_1, \ldots, \phi_k\}$ of vertex-disjoint contigs such that either $k = 1$ or ϕ_i is linear for every $1 \le i \le k$. A *straight representation* is a round representation whose contigs are all linear. We extend the notation used for contigs to round representations:

- $\mathcal{B}(\Phi) = \bigcup_{1\le i\le k} \mathcal{B}(\phi_i)$ and $V(\Phi) = \bigcup_{1\le i\le k} V(\phi_i)$,
- if $B \in \mathcal{B}(\phi_i)$, then $f^\Phi(B) = f^{\phi_i}(B)$ for every $f \in \{L, R, F_l, F_r, U_l, U_r\}$,
- if $W \in \mathcal{B}(\phi_j)$, then $B \longrightarrow_\Phi W$ if and only if $i = j$ and $B \longrightarrow_{\phi_i} W$, and
- $B \not\longrightarrow_\Phi W$ if and only if $j \neq i$ or $B \not\longrightarrow_{\phi_i} W$.

As usual, we omit Φ from the previous notation. Note that Φ is uniquely determined by the triplet $\langle \mathcal{B}(\Phi), R, F_r\rangle$, thus sometimes we write $\langle \mathcal{B}(\Phi), R, F_r\rangle$ as an alternative definition of Φ. (Note that $\mathcal{B}(\Phi)$ is a family of semiblocks and not a family of orderings.) Any (linear) contig ϕ can be regarded as the (straight) round representation $\{\phi\} = \langle \mathcal{B}(\phi), R^\phi, F_r^\phi\rangle$, thus all the definitions that follow for round representations hold for contigs as well. We say that a semiblock $B \in \mathcal{B}(\Phi)$ is a *left (resp. right) end* semiblock when $F_l(B) = B$ (resp. $F_r(B) = B$). Equivalently, B is a left (resp. right) semiblock of Φ if and only if B is the leftmost (resp. rightmost) of its contig, which happens if and only if $L(B) = \bot$ (resp. $R(B) = \bot$). We treat $\bot$ as a special semiblock outside $\mathcal{B}(\Phi)$, one for which $f(\bot) = \bot$ for every $f \in \{L, R, F_r, F_l\}$. In Figure 1, $\Phi = \{\phi, \psi\}$ and $\Gamma = \{\gamma\}$ are round representations, whereas $\{\phi, \gamma\}$ is not. Moreover, B_1 and B_6 are the left end semiblocks of Φ, while γ has no end semiblocks. Indeed, a round representation is straight if and only if it contains end semiblocks.

Each round representation Φ defines a graph $G(\Phi)$ that has $V(\Phi)$ as it vertex set, where $v \in B$ and $w \in W$ are adjacent, for $B, W \in \mathcal{B}(\Phi)$, if and only if $B \overset{=}{\longrightarrow} W$, or $W \overset{=}{\longrightarrow} B$. Observe that: each contig $\phi \in \Phi$ defines a component $G(\phi)$ of $G(\Phi)$, each semiblock of Φ is a semiblock

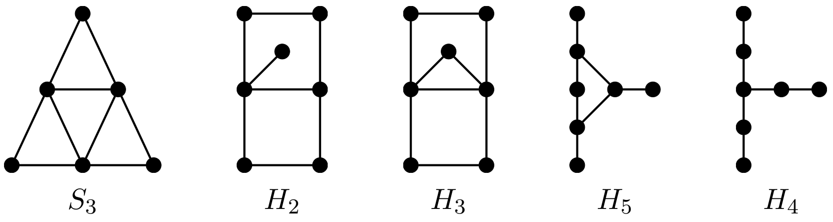


Figure 2: Complements of the forbidden induced subgraphs for PCA graphs

of $G(\Phi)$, and $N_{G(\Phi)}[v] = \bigcup[F_l(B), F_r(B)]$ for every vertex $v \in B$ of Φ. We say a semiblock of $\mathcal{B}(\Phi)$ is *isolated* or *universal* according to whether it is isolated or universal in $G(\Phi)$. Similarly, two semiblocks of $\mathcal{B}(\Phi)$ are adjacent or twins when they are adjacent or twins in Φ. We write $N_\Phi(B)$ to denote the set of semiblocks adjacent to B and $N_\Phi[B] = N_\Phi(B) \cup \{B\}$; we drop the subindex Φ as usual. Note that $N[B] = [F_l(B), F_r(B)]$. In Figure 1, $G(\{\gamma\})$ is obtained from a cycle with four vertices w_1, w_2, w_3, w_4 by adding $|W_i| - 1$ twins of w_i, for $1 \leq i \leq 4$. Also, B_3 is universal in $\{\phi\}$ but not in $\{\phi, \psi\}$.

A graph G is a *proper circular-arc (PCA)* graph if it is isomorphic to $G(\Phi)$ for some round representation Φ. In such a case, G *admits* Φ, while Φ *represents* G. It is a well known fact that PCA graphs are precisely those graphs that admit a PCA model, as defined in Section 1 [8, 16, 30]. PCA graphs are characterized by a family of minimal forbidden induced subgraphs, as in Theorem 2.1. There, H^* denotes the graph that is obtained from H by inserting an isolated vertex, while C_n denotes the cycle with n vertices. Graph $\overline{C_3^*}$ is also denoted by $K_{1,3}$.

Theorem 2.1 ([32])**.** *A graph is a PCA graph if and only if it does not contain as induced subgraphs any of the following graphs: C_n^* for $n \geq 4$, $\overline{C_{2n}}$ for $n \geq 3$, $\overline{C_{2n+1}^*}$ for $n \geq 1$, and the graphs $\overline{S_3}$, $\overline{H_2}$, $\overline{H_3}$, $\overline{H_4}$, $\overline{H_5}$ and S_3^* (see Figure 2).*

Proper interval graphs are defined as PCA graphs, by replacing round representations with straight representations. That is, a graph is a *proper interval graph (PIG)* graph when it is isomorphic to $G(\Phi)$ for some straight representation Φ. It is well known that PIG graphs are precisely those graphs that admit PIG models [8, 13]. PIG graphs are also characterized by minimal forbidden induced subgraphs.

Theorem 2.2 ([22])**.** *A PCA graph is a PIG graph if and only if it does not contain C_k for $k \geq 4$, and S_3 as induced subgraphs.*

Two semiblocks B, W of a round representation Φ are *indistinguishable* when $F_l(B) = F_l(W)$ and $F_r(B) = F_r(W)$ (e.g., B_7 and B_8 in Figure 1). Clearly, if $B \longrightarrow W$, then all the semiblocks in $[B, W]$ are pairwise indistinguishable in Φ. It is not hard to see that B and W are twins when they are indistinguishable. We say that Φ and a round representation Ψ are *equal* when Φ can be obtained from Ψ by permuting some indistinguishable semiblocks in the contigs of Ψ. Of course, Ψ is an alternative round representation of $G(\Phi)$. A PCA graph can also have non-equal representations. Indeed, $\Phi^{-1} = \langle \mathcal{B}(\Phi), L^\Phi, F_l^\Phi \rangle$, which is called the *reverse* of Φ, is a representation of $G(\Phi)$. By definition, $R^{\Phi^{-1}} = L^\Phi$, $F_l^{\Phi^{-1}} = F_r^\Phi$, and $\Phi^{-1} = \{\phi^{-1} \mid \phi \in \Phi\}$.

For $\mathcal{B} \subseteq \mathcal{B}(\Phi)$, we write $\Phi|\mathcal{B}$ to denote the round representation Ψ such that $\mathcal{B}(\Psi) = \mathcal{B}$ and $B \longrightarrow_\Psi W$ if and only if $B \longrightarrow_\Phi W$ for every $B, W \in \mathcal{B}$. Observe that Ψ is a round representation of $G(\Phi)[V(\Psi)]$, thus Ψ is referred to as the representation of Φ *induced by* $\mathcal{B}$. Similarly, the *removal of* $\mathcal{B}$ from Φ is the representation $\Phi \setminus \mathcal{B} = \Phi|(\mathcal{B}(\Phi) \setminus \mathcal{B})$; this time, $G(\Phi \setminus \mathcal{B}) = G(\Phi) \setminus (\bigcup \mathcal{B})$.

We extend the notion of ranges to round representations. Let B_i be a semiblock of a contig ϕ_i of the round representation Φ, for $i \in \{1, 2\}$. When $\phi_1 = \phi_2$, the *range* $[B_1, B_2]$ of Φ is defined as the range $[B_1, B_2]$ of $\mathcal{B}(\phi_1)$. When $\phi_1 \neq \phi_2$, the *range* $[B_1, B_2]$ of Φ is defined as the range $[B_1, B_2]$ of $\mathcal{B}(\phi_1) \bullet \mathcal{B}(\phi_2)$. That is, $[B_1, B_2]$ is obtained by traversing ϕ_1 from B_1 to its right end semiblock, followed by the range obtained by traversing ϕ_2 from its left end semiblock to B_2. This non-standard definition is useful to deal with the different possible orderings of the contigs of Φ; in this case, ϕ_1 would immediately precede ϕ_2. For instance, $[B_4, B_7]$ of $\{\phi, \psi\}$ in Figure 1 is B_4, B_5, B_6, B_7. The ranges $(B, W]$, $[B, W)$, and (B, W) of Φ are defined analogously.

Our definition of ranges allows us to define some robust versions of L, R, U_l, and U_r. By definition, any range $\mathcal{B} = [B_1, B_2]$ of Φ is a linear ordering, thus $R^{\mathcal{B}}(B)$ is the semiblock that follows B in $\mathcal{B}$, for any $B \in [B_1, B_2)$. Let ϕ_1 and ϕ_2 be the contigs that contain B_1 and B_2, respectively. By definition, $\mathcal{B}$ could contain the right end semiblock B_r of ϕ_1 followed by the left end semiblock B_l of ϕ_2. Although $R^{\Phi}(B_r) = \bot$ and $L^{\Phi}(B_l) = \bot$, the semiblocks $R^{\mathcal{B}}(B_r) = B_l$ and $L^{\mathcal{B}}(B_l) = B_r$ are well defined. The *robust* version $R^{\langle\Phi,\mathcal{B}\rangle}$ of R^{Φ} behaves exactly as R^{Φ}, with the exception that it maps B_r to B_l. Similarly, the *robust* version $L^{\langle\Phi,\mathcal{B}\rangle}$ of L^{Φ} maps B_l to B_r and B to $L^{\Phi}(B)$ for $B \neq \mathcal{B}(\Phi) \setminus \{B_l\}$. The *robust* versions $U_l^{\langle\Phi,\mathcal{B}\rangle}$ of U_l^{Φ}, and $U_r^{\langle\Phi,\mathcal{B}\rangle}$ of U_r^{Φ} are defined analogously. For the sake of notation, we write $\hat{L}$, $\hat{R}$, $\hat{U}_l$ and $\hat{U}_r$ when Φ and $\mathcal{B}$ are clear. Thus, if we consider the range $[B_4, B_7]$ of $\{\phi, \psi\}$ in Figure 1, then $\hat{R}(B_5) = B_6$, $\hat{L}(B_6) = B_5$, thus $\hat{U}_r(B_3) = B_6$, and $\hat{U}_l(B_8) = B_5$; note, however, that $\hat{L}(B_1) = \hat{R}(B_{10}) = \bot$.

Before we advance, we describe the rationale behind our definitions of round representations, ranges, and the robust mappings $\hat{L}$ and $\hat{R}$. Our main goal in this article is to insert a new vertex v into a round representation Φ to obtain a new round representation Ψ. As we shall see, v can have neighbors in at most two contigs ϕ_1 and ϕ_2 of Φ (possibly $\phi_1 = \phi_2$). To insert v, we must join the semiblocks in $\mathcal{B}(\phi_1) \cup \mathcal{B}(\phi_2)$ together with v into a new contig ψ that "replaces" both ϕ_1 and ϕ_2, i.e., $\Psi = (\Phi \setminus \{\phi_1, \phi_2\}) \cup \{\psi\}$. Since ψ is a contig, $\mathcal{B}(\phi_1) \cup \mathcal{B}(\phi_2)$ must be somehow together in Ψ. Prior to the insertion of v, any pair of contigs of Φ could play the role of ϕ_1 and ϕ_2, thus it is inconvenient to prefix an ordering of the contigs of Φ. As this ordering is absent, it makes no sense to define the follower (resp. predecessor) of a right (resp. left) end semiblock. However, once v and $N(v)$ are given, we have access to the neighbor semiblocks in ϕ_1 and ϕ_2. A priori, there is no way of knowing if $\phi_1 = \phi_2$; all we can query is if v is adjacent to end semiblocks. Yet, since ψ is a contig, the semiblocks adjacent to v must appear consecutively in ψ. In other words, $N(v)$ should be a range of $[B_1, B_2]$ of $\mathcal{B}(\phi_1) \bullet \mathcal{B}(\phi_2)$. We want to make no case distinctions according to whether $\phi_1 = \phi_2$ or whether $[B_1, B_2]$ has end semiblocks. This is the reason why ranges are defined for semiblocks in different contigs, and why the range of an ordering can include its rightmost element. Finally, to test if v can be inserted, we have to check some conditions on $R(B_m)$ for $B_m \in [B_1, B_2]$. However, this semiblock is not well defined when $R(B_m) = \bot$ and, in this case, the role of this semiblock is played by the left end semiblock of ϕ_2. The robust definition of $\hat{R}$ allows us to treat the case in which $R(B_m) = \bot$ in the same way as we treat the other case.

Although we need access to $\mathcal{B}$ for the robust versions to be efficient, there is one case in which specifying $\mathcal{B}$ is not required. If $\Phi = \{\phi_1, \phi_2\}$ for (possibly equal) contigs ϕ_1, ϕ_2, then $\hat{R}(B_r) = B_l$ and $\hat{L}(B_l) = B_r$ for the left end semiblock B_l of ϕ_i and the right end semiblock B_r of ϕ_j (if existing), $i, j = \{1, 2\}$, while $\hat{R}(B) = R(B)$ and $\hat{L}(B) = L(B)$ for every other semiblock. In this case, we refer to Φ is being *robust*.

By definition, each contig ϕ of a straight representation Φ is "equivalent" to a range $[B_l, B_r]$ of Φ, where B_l is a left end semiblock, B_r is a right end semiblock, and (B_l, B_r) has no end semiblocks. The term "equivalent" employed here means that $\{\phi\}$ equals $\Phi|[B_l, B_r]$; moreover, $\Phi|[B_l, B_r]$ is a round representation of some component of G' of $G(\Phi)$. We refer to $[B_l, B_r]$ simply as a *contig range* of Φ that *describes* G'. The following observation then follows.

Observation 2.3. *If Φ is a straight representation of a graph G, then every component of G is described by a contig range.*

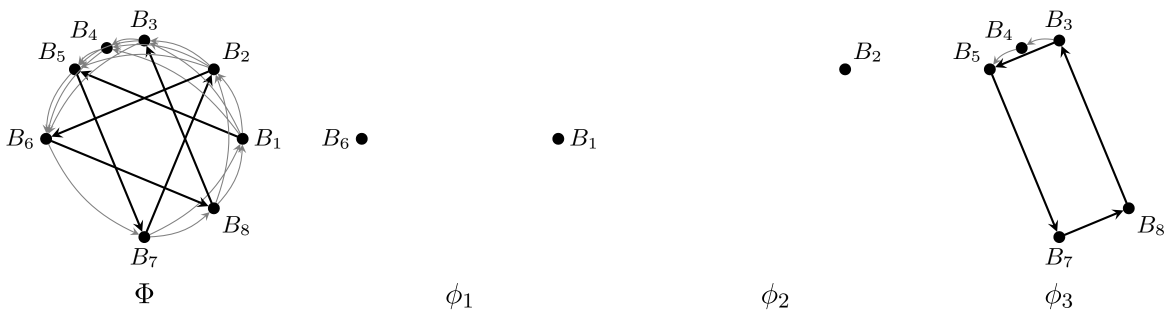


Figure 3: A round representation Φ with three co-contigs ϕ_1, ϕ_2, and ϕ_3 (gray lines are used to depict the edges that are implied by black edges). The left co-end semiblocks of Φ are B_1, B_2, B_3, B_6, and B_7, while its right co-end semiblocks are B_1, B_2, B_5, B_6, and B_8. Note that each co-contig pair other than $\langle B_2, \emptyset\rangle$ describes a co-component of $G(\Phi)$.

In a similar way, the co-components of co-bipartite PCA graphs are described by co-contig pairs (see Figure 3). Let Φ be a round representation of a co-bipartite graph G; observe Φ is robust. Say that a non-universal semiblock $B \in \mathcal{B}(\Phi)$ is a *left co-end semiblock* of Φ if $B = \hat{U}_r^2(B)$; similarly, B is a *right co-end semiblock* of Φ when $B = \hat{U}_l^2(B)$. A *co-contig range* is a range $[B_l, B_r]$ such that B_l is a left co-end semiblock, B_r is a right co-end semiblock, and (B_l, B_r) has no co-end semiblocks. As it is shown in [16, 30], $\overline{\mathcal{B}} = [\hat{U}_r(B_l), \hat{U}_l(B_r)]$ is also a co-contig range for any co-contig range $\mathcal{B} = [B_l, B_r]$. Moreover, $G' = G[\bigcup(\mathcal{B} \cup \overline{\mathcal{B}})]$ is a co-component of G, and $\langle \bigcup \mathcal{B}, \bigcup \overline{\mathcal{B}}\rangle$ is a co-bipartition of G'. The pair $\langle \mathcal{B}, \overline{\mathcal{B}}\rangle$ is said to be a *co-contig pair* of Φ that *describes* G', while $\Phi|(\mathcal{B} \cup \overline{\mathcal{B}})$ is a *co-contig* of Φ.

Theorem 2.4 ([16, 30])**.** *If Φ is a round representation of a co-bipartite graph G, then Φ is robust and every co-component of G with at least two vertices is described by a co-contig pair.*

Our definition of co-contig pairs above explicitly excludes universal semiblocks. Clearly, each vertex in a universal semiblock induces a co-component of G. We say that a universal semiblock B is both a *left co-end* and *right co-end* semiblock. Hence, $\mathcal{B} = [B, B]$ is a *co-contig range*, $\langle \mathcal{B}, \emptyset\rangle$ is a *co-contig pair* that describes $G' = G[\bigcup \mathcal{B}]$, and $\Phi|\{B\}$ is a *co-contig* of Φ.

As defined so far, co-contigs only represent co-bipartite graphs. For the sake of generality, we say that a round representation Φ is a *co-contig* of Φ when $G(\Phi)$ is co-connected. Note that, consequently, we may not assume that co-contigs are robust or have co-contig ranges. Also, to make clear the parallelism between contigs and co-contigs, we use lowercase Greek letters to name the co-contigs of Φ when $G(\Phi)$ needs not be co-connected.

3 The Reception Theorem: a certification roadmap

Receptive pairs are the main concept used in [30] for dealing with the insertion of a non-isolated vertex v into G. In simple terms, a pair of semiblocks is receptive when it witnesses that $H = G \cup \{v\}$ is PCA. Its definition, however, depends on whether v belongs to an end semiblock or not. Suppose H is represented by a round representation Ψ and $\{v\}$ is a semiblock of Ψ. Let $B_l = F_l(\{v\})$ if $\{v\}$ is not a left end semiblock, and $B_l = R(\{v\})$ otherwise. Similarly, $B_r = F_r(\{v\})$ if $\{v\}$ is not a right end semiblock, while $B_r = L(\{v\})$ otherwise. By definition, $\Phi = \Psi \setminus \{v\}$ is a round representation of G. The pair $\langle B_l, B_r\rangle$ is referred to as being *receptive* in Φ, while Ψ is the *$\{v\}$-reception of $\langle B_l, B_r\rangle$* in Φ. Strictly speaking, v plays no role when deciding if a pair is receptive; $\langle B_l, B_r\rangle$ is receptive if and only if $G \cup \{w\}$ is a PCA graph for any w with $N(w) = N(v) = \bigcup[B_l, B_r]$ (recall v is not isolated). When applied to H and v, we

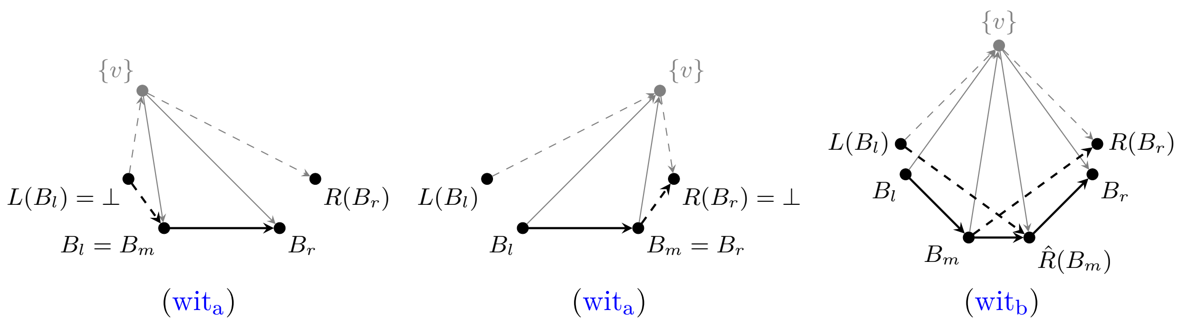


Figure 4: The possibilities for a semiblock B_m to witness that $\langle B_l, B_r \rangle$ is receptive. A dashed edge from X to Y means that $X \not\longrightarrow Y$, while $X \longrightarrow Y$ is possible when the edge is missing (and Φ is a round representation). Also, some of the depicted semiblocks could be equal. The corresponding $\{v\}$-receptions are drawn in gray.

obtain that H is a PCA graph if and only if G admits a round representation Φ with a receptive pair $\langle B_l, B_r \rangle$ such that $N(v) = \bigcup[B_l, B_r]$.

As defined, the concept of receptive pairs applies to any round representation. Yet, the dynamic algorithm deals with a rather restricted subset of round representations. Say that a semiblock B of a round representation Ψ is a *block* when B is a block of $G(\Psi)$. If every semiblock of Ψ is a block, then Ψ is a *round block representation* and all its (co-)contigs are referred to as *block (co-)contigs*. When Ψ is a round block representation, the round representation $\Phi = \Psi \setminus \{v\}$ is almost a block representation. In fact, it can be easily observed that $\{L(B_l), B_l\}$ and $\{B_r, R(B_r)\}$ are the unique possible pairs of indistinguishable semiblocks of Φ, while Φ has at most two universal semiblocks, one in $[B_l, B_r]$ and the other outside $[B_l, B_r]$. For the sake of notation, we refer to Φ as *v-receptive* when it contains a receptive pair $\langle B_l, B_r \rangle$ such that:

- $N(v) = \bigcup[B_l, B_r]$, and
- no pair of semiblocks in $\mathcal{B}(\Phi) \setminus \{B_l, B_r\}$ are indistinguishable.

Theorem 3.1. *Let H be a graph such that $v \in V(H)$ is not isolated. Then, H is PCA if and only if $H \setminus \{v\}$ admits a round representation Φ that is v-receptive. Furthermore, the $\{v\}$-reception Φ is a round representation of H.*

The above observation is quite straightforward, but it tells us little about the v-receptive representations of G. In [8, 13, 30], tools for efficiently finding and transforming Φ into a round representation of H are developed. The Receptive Pair Lemma of [30], that generalizes some results in [8, 13], is one of such tools. For the sake of simplicity, we present a unified view of [8, 13, 30].

Let Φ be a round representation and $B_l \neq B_r$ be semiblocks of $\mathcal{B}(\Phi)$. A semiblock $B_m \in \mathcal{B}(\Phi)$ *witnesses that $\langle B_l, B_r \rangle$ is receptive* in Φ when (see Figure 4):

(wit$_a$) B_m is an end semiblock, $B_l \overset{=}{\longrightarrow} B_m$, and $B_m \overset{=}{\longrightarrow} B_r$, or

(wit$_b$) $B_l \overset{=}{\longrightarrow} B_m$, $B_m \not\longrightarrow R(B_r)$, $L(B_l) \not\longrightarrow R(B_m)$, and $\hat{R}(B_m) \overset{=}{\longrightarrow} B_r$.

The essence of the insertions methods in [8, 13, 30] is captured in the next lemma.

Lemma 3.2 (Receptive Pair Lemma [8, 13, 30])**.** *Let ϕ_1, ϕ_2 be two (possibly equal) contigs that contain the semiblocks B_l and B_r, respectively. Then, $\langle B_l, B_r \rangle$ is receptive in $\{\phi_1, \phi_2\}$ if and only if $B_m \in [B_l, B_r]$ witnesses that $\langle B_l, B_r \rangle$ is receptive in $\{\phi_1, \phi_2\}$.*

The Receptive Pair Lemma can be proved with not much effort by following the definitions (see [30] and Figure 4). Indeed, if B_m witnesses that $\langle B_l, B_r\rangle$ is receptive, then a $\{v\}$-reception is obtained by: inserting $\{v\}$ immediately to the right of B_m if (wit$_\text{b}$) or $B_m \neq B_l$; or inserting $\{v\}$ immediately to the left of $B_m = B_l$ if (wit$_\text{a}$). On the other hand, if Ψ is a $\{v\}$-reception of $\langle B_l, B_r\rangle$, then either $L(\{v\})$ or $R(\{v\})$ (if $L(\{v\}) = \bot$) witnesses that $\langle B_l, B_r\rangle$ is receptive. The Receptive Pair Lemma is an asymmetric tool: it suffices to find one v-receptive representation of G to claim that H is PCA, while all the round representations of G must be discarded before claiming that H is not PCA. Moreover, a round representation of H is available when H is PCA, whereas no minimally forbidden is found when H is not PCA. The Reception Theorem combines Theorem 3.1 with a slight generalization of the Receptive Pair Lemma that takes $N(v)$ into account. For a better exposition, we consider only the case in which H is connected. Nevertheless, conditions (rec$_1$)–(rec$_3$) are general.

Corollary 3.3 (Reception Theorem)**.** *Let H be a connected graph with a vertex v. Then, H is PCA if and only if $H \setminus \{v\}$ admits a round representation Φ that contains two semiblocks B_l, B_r such that:*

(rec$_1$) $N(v) = \bigcup[B_l, B_r]$,

(rec$_2$) no pair of semiblocks in $\Phi \setminus \{B_l, B_r\}$ are indistinguishable,

(rec$_3$) $B_m \in [B_l, B_r]$ witnesses that $\langle B_l, B_r\rangle$ is receptive in Φ.

Technically speaking, (rec$_1$)–(rec$_3$) are statements dealing with pairs of semiblocks. For the sake of simplicity, we say that a round representation Φ *satisfies* a subset P of (rec$_1$)–(rec$_3$) when Φ has two semiblocks B_l and B_r that simultaneously satisfies all the conditions in P.

Despite the simplicity of the Reception Theorem, the problem of finding a v-receptive representation is not an easy task, specially when the time constraints are tight. Most of the effort in [8, 13, 30] is spent on finding such a v-receptive representation efficiently. The problem of finding a minimally forbidden is mostly, but not completely, ignored in these articles. In fact, the Reception Theorem made its first appearance in [8], where the authors consider a rather restricted case in which G is PIG and both G and H are connected. The advantage of this case is that G admits only two contigs, namely γ and γ^{-1}. By (rec$_1$), $N(v)$ must be a range of γ, which implies that there are exactly two contigs ϕ and ϕ^{-1} that can satisfy (rec$_1$) and (rec$_2$). In their proof of the Reception Theorem, Deng et al. [8] exhibit a minimally forbidden of H when either $N(v)$ is not consecutive in γ or ϕ does not satisfy (rec$_3$). Although it is not explicit in [8], an $O(n)$ time algorithm for obtaining such a minimally forbidden, when γ and $N(v)$ are give as input, follows as a by-product. It is not hard to extend this certified algorithm to the case in which the PIG graph G can be disconnected.

Our aim in the present manuscript is to design a certifying and dynamic algorithm for the recognition of PCA graphs, following the framework discussed in Section 1. Thus, we ought to compute a minimally forbidden each time an insertion of a vertex v fails. The main idea is to prove the Reception Theorem following the same path as Deng et al. That is, we show a minimally forbidden of H when **no** round representation of $H \setminus \{v\}$ is v-receptive. However, we spend $O(d(v) + \text{ep}^+)$ time to build the minimally forbidden.

4 The data structure

As anticipated, the algorithm keeps two differentiated data structures. One implements a round block representation to witness that G is PCA, while the other represents an induced path of G that is extended to a negative witness when the insertion of a vertex fails. The following sections present the data types involved in the dynamic algorithm.

4.1 Contigs

The data structure we use to implement contigs is the same as discussed in [30]. For completeness, we describe its interface as an abstract data type; for implementations details see [30].

Each contig ϕ presents itself as the collection of semiblocks $\mathcal{B}(\phi)$, where each $B \in \mathcal{B}(\phi)$ stores its set of vertices. Also, each $B \in \mathcal{B}(\phi)$ and each $v \in B$ allow the user to store some additional data. The internal structure and the semiblocks and vertices of ϕ are exclusively owned by ϕ, thus the modifications applied on ϕ have no impact on the data structures of other contigs. Moreover, a user cannot directly access ϕ, a semiblock $B \in \mathcal{B}(\phi)$, or a vertex $v \in B$. Instead, a *semiblock (resp. vertex) pointer* $\tilde{B}$ (resp. $\tilde{v}$) *associated to* ϕ must be employed to access B (resp. v). For simplicity, we also say that ϕ is *referenced by* $\tilde{B}$ (resp. $\tilde{v}$). The pointer $\tilde{B}$ is aware of the internal structure of ϕ, thus it can be used used to manipulate both B and ϕ. However, $\tilde{B}$ knows nothing about the other semiblock pointers associated to ϕ or the semiblocks in $\mathcal{B}(\phi) \setminus \{B\}$. Hence, there is no way to answer, in $O(1)$ time, if $\tilde{B}$ is associated to the same contig as other pointer $\tilde{W}$. (Roughly speaking, ϕ is similar to doubly linked lists in which the semiblocks play the role of nodes and semiblock pointers are pointers to the nodes.)

The following functions that operate on contigs and semiblocks are provided in [8, 13, 30]. As usual, we use lower case Greek letters for contigs, capital Roman letters for semiblocks, and tildes to indicate pointers.

`newContig()` creates a new contig that contains only one block $B = \{v\}$ and returns the pointers to B and v. Complexity: $O(1)$ time.

`vertices(`$\tilde{B}$`)` returns (an iterator to) $\{\tilde{v} \mid v \in B\}$. Complexity: $O(1)$ time.

`semiblock(`$\tilde{v}$`)` returns a pointer to the semiblock that contains v. Complexity $O(1)$ time.

$f(\tilde{B})$ returns a semiblock pointer to $f(B)$ for $f \in \{L, R, F_l, F_r, U_r, U_l\}$. Complexity: $O(1)$ time.

$[\tilde{B}_1, \tilde{B}_2]$ returns a list of semiblocks pointers that represents the range $[B_1, B_2]$ of $\{\phi_1, \phi_2\}$, where ϕ_i is the contig referenced by $\tilde{B}_i$ for $i \in \{1, 2\}$. The ranges $(\tilde{B}_1, \tilde{B}_2]$, $[\tilde{B}_1, \tilde{B}_2)$ and $(\tilde{B}_1, \tilde{B}_2)$ work similarly. Complexity: $O(\#[B_1, B_2])$ time.

`receptive(`$\tilde{B}_l$`, `$\tilde{B}_r$`)` is true when $\langle B_l, B_r \rangle$ is receptive in $\{\phi_1, \phi_2\}$, where ϕ_1 and ϕ_2 are the contigs referenced by $\tilde{B}_l$ and $\tilde{B}_r$, respectively. Complexity: $O(\#[B_l, B_r])$ time.

`reception(`$\tilde{B}_l$`, `$\tilde{B}_r$`)` transforms ϕ_1 and ϕ_2 into the $\{v\}$-reception ψ of $\langle B_l, B_r \rangle$, where ϕ_1 and ϕ_2 are the contigs referenced by $\tilde{B}_l$ and $\tilde{B}_r$, respectively. Returns a pointer to $\{v\} \in \mathcal{B}(\psi)$. Requires $\langle B_l, B_r \rangle$ to be receptive in $\{\phi_1, \phi_2\}$. Complexity: $O(\#[B_l, B_r])$ time.

`remove(`$\tilde{B}$`)` transforms the contig ψ referenced by $\tilde{B}$ into the contigs of $\{\psi\} \setminus \{B\}$ and the contig γ whose only semiblock is B. Note that $\{\psi\} \setminus \{B\}$ has either one or two contigs, thus at most three contigs are generated. Complexity: $O(\#[F_l(B), F_r(B)])$ time.

`separate(`$\tilde{B}$`, `W`)` transforms the contig γ referenced by $\tilde{B}$ into a contig ϕ representing $G(\gamma)$ that is obtained by splitting B into two indistinguishable semiblocks $B \setminus W$ and W in such a way that $R^{\phi}(W) = R^{\gamma}(B)$, $L^{\phi}(W) = B \setminus W$, and $L^{\phi}(B \setminus W) = L^{\gamma}(B)$. The other semiblocks of γ are not affected by this operation. It requires $W \subseteq B$, and it has no effects when either $W = B$ or $W = \emptyset$. Note that W is not a semiblock pointer, but a set of vertex pointers. Complexity: $O(|W|)$ time. (See Figure 5.)

`separate(`W`, `$\tilde{B}$`)` does the same as `separate(`$\tilde{B}$`, `$B \setminus W$`)`. Complexity: $O(|W|)$ time.

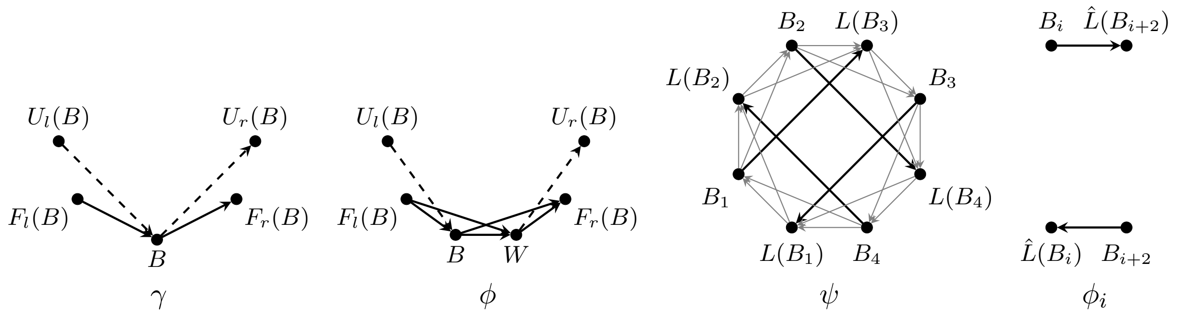


Figure 5: Effects of some operations on contigs: γ equals `compact(`$\tilde{B}$`)` for $B \in \mathcal{B}(\phi)$; ϕ equals `separate(`$\tilde{B}$`, `W`)` for $B \in \mathcal{B}(\gamma)$; ψ equals `join(`$\tilde{B}_1$`, `$\tilde{B}_2$`)` for $B_1 \in \phi_1$ and $B_2 \in \phi_2$; and $\{\phi_i, \phi_2\}$ equals `split(`$\tilde{B}_1$`, `$\tilde{B}_2$`)` for $B_1, B_2 \in \psi$. Dashed lines represent non-edges, while missing lines represent edges that could be present of absent. Note that some blocks may be equal, while B_3 (resp. B_4) does not exist when B_1 (resp. B_2) is universal.

`compact(`$\tilde{B}$`)` transforms the contig ϕ referenced by $\tilde{B}$ into a contig γ representing $G(\phi)$ that is obtained by joining the indistinguishable semiblocks B and $R(B)$ into one semiblock $B \cup R(B)$ in such a way that $L^\gamma(B \cup R(B)) = L^\phi(B)$ and $R^\gamma(B \cup R(B)) = R^\phi(B)$. The other semiblocks of ϕ are not affected by this operation. It has no effects when B and $R(B)$ are not indistinguishable. Complexity: $O(\min(|B|, |R(B)|)$ time. (See Figure 5.)

`join(`$\tilde{B}_1$`, `$\tilde{B}_2$`)` transforms ϕ_1 and ϕ_2 into a new contig ψ that represents $G(\phi_1) + G(\phi_2)$, where ϕ_i is the contig referenced by $\tilde{B}_i$ for $i \in \{1, 2\}$. The semiblocks of ψ appear as in the ordering $[B_1, F_r^{\phi_1}(B_1)] \bullet [B_2, F_r^{\phi_2}(B_2)] \bullet (F_r^{\phi_1}(B_1), B_1) \bullet (F_r^{\phi_2}(B_2), B_2)$. It has an undefined behavior when either 1. B_1 or B_2 is not a left co-end block or 2. $\tilde{B}_1$ and $\tilde{B}_2$ are associated to the same contig. Time complexity: $O(u)$ time, where u is the number of universal semiblocks in ψ. (See Figure 5.)

`split(`$\tilde{B}_1$`, `$\tilde{B}_2$`)` transforms the contig ψ referenced by $\tilde{B}_1$ and $\tilde{B}_2$ into two co-contigs ϕ_1 and ϕ_2 in such a way that $G(\psi) = G(\phi_1) + G(\phi_2)$. The semiblocks of ϕ_1 appear as in the ordering $[B_1, B_2) \bullet (F_r^\psi(B_1), F_r^\psi(B_2)]$, while the semiblocks of ϕ_2 appear as in the ordering $[B_2, F_r^\psi(B_1)] \bullet (F_r^\psi(B_2), B_1)$. It has an undefined behavior when either 1. B_1 or B_2 is not a left co-end block or 2. $\tilde{B}_1$ and $\tilde{B}_2$ are associated to different contigs. Note that this operation requires $B_1 \longrightarrow B_2$; if this is not the case, then $B_2 \longrightarrow B_1$, thus the operation works as if the parameters where inverted. Time complexity: $O(u)$ time, where u is the number of universal semiblocks in ψ. (See Figure 5.)

Several of the above operations *transform* a contig ϕ into a new contig ψ. This means that the physical structure of ϕ is changed to obtain ψ, thus ϕ ceases to exist as such. The semiblock pointers associated to ϕ are not invalidated, though; instead, they get associated to ψ. We remark, also, that `split(`B_1, B_2`)` reverses the effects of `join(`B_1`, `B_2`)`, but the converse is not true in general. The subtle problem is that `split` outputs co-contigs, and co-contigs involve two contigs when they represents disconnected graphs (see Figure 3 (ϕ_1) and Figure 5 (ϕ_i)). We deal with this problem in Section 4.3.

4.2 Semiblock paths

Together with each contig ϕ, the dynamic algorithm keeps a path of ϕ that is used for the generation of negative witnesses. Say that a semiblock $B \in \mathcal{B}(\phi)$ is *long* when $F_r(B) \longrightarrow F_l(B)$;

those semiblocks that are not long are referred to as *short*. A *semiblock path* $\mathcal{P}$ of ϕ is an ordering $B_1, \ldots, B_k$ of a subset of $\mathcal{B}(\phi)$ such that:

- $B_i \longrightarrow B_{i+1}$ and $B_i \not\longrightarrow B_j$ for every $1 \leq i < k$ and $j \neq i+1$.
- If ϕ is linear, then $\mathcal{P}$ is linear and B_1 and B_k are the end semiblocks of ϕ.
- If ϕ is circular, then $\mathcal{P}$ is circular and $B_k \longrightarrow B_1$ and $B_k \not\longrightarrow B_2$.
- If ϕ contains a long semiblock, then $k = 3$.

Each semiblock path $\mathcal{P}(\phi) = B_1, \ldots, B_k$ is implemented with a doubly linked list of semiblock pointers $\tilde{B}_1, \ldots, \tilde{B}_k$; the list is circular if and only if $\mathcal{P}(\phi)$ is circular. Conversely, each semiblock $B \in \mathcal{B}(\phi)$ has a *path pointer* to the position that $\tilde{B}$ occupies in $\mathcal{P}(\phi)$; this is a null value when $B \notin \mathcal{P}(\phi)$. Thus, $O(1)$ time is enough to detect if B belongs $\mathcal{P}(\phi)$, to access and remove $\tilde{B}$ from $\mathcal{P}(\phi)$, and to insert new semiblock pointers in $\mathcal{P}(\phi)$ to the left or the right of $\tilde{B}$.

We now show how to efficiently update $\mathcal{P}(\phi)$ (and the path pointers of ϕ) each time a contig ϕ is updated. As discussed in the previous section, there are eight operations that change the structure of a contig: `newContig`, `leftSeparate`, `rightSeparate`, `compact`, `join`, `split`, `reception`, and `remove`. Updating $\mathcal{P}(\phi)$ in $O(1)$ time is trivial for the first four operations. Regarding `join` and `split`, note that the input and output contigs represent co-bipartite graphs. Thus, the semiblock paths of the input contigs can be erased in $O(1)$ time because they have at most four semiblock pointers. After the semiblock paths are erased, we build the new semiblock paths from scratch as in the next lemma.

Lemma 4.1. *Let B_l be a left co-end semiblock of a contig ϕ. If a semiblock pointer to B_l is given, then a semiblock path can be computed in $O(1)$ time.*

Proof. First suppose ϕ is a linear contig and observe that, in this case, $W_l = F_l^3(B)$ is the left end semiblock of ϕ. Indeed, if W_l is not a left end semiblock, then $F_l(W_l)$, $F_l^2(B_l)$, and B_l are pairwise non-adjacent semiblocks, which contradicts the fact that $G(\phi)$ is co-bipartite. Similarly, $F_r^3(W_l)$ is the right end semiblock of ϕ. Thus, $F_r^0(W_l), \ldots, F_r^i(W_l)$ is a semiblock path, where $0 \leq i \leq 3$ is the minimum such that $F_r^i(W_l)$ is a right co-end semiblock. Clearly, this semiblock path can be computed in $O(1)$ time.

Now suppose ϕ is circular and let $B_r = F_r(B_l)$, $W_l = U_r(B_l)$ and $W_r = F_r(W_l)$. Recall that W_l is a left co-end block while B_r and W_r are right co-end blocks. If $F_l(B_l) = W_l$, then B_l is long and B_l, B_r, W_l is a semiblock path. Otherwise, $[B_l, B_r]$ and $[W_l, W_r]$ is a partition of $\mathcal{B}(\phi)$. Moreover, $B_r \longrightarrow W_l$ and $W_r \longrightarrow B_l$ because ϕ is circular. We consider two cases:

Case 1: ϕ contains a semiblock path B_1, B_2, B_3. Note that at least one of B_1, B_2, B_3 belongs to $[B_l, B_r]$ (resp. $[W_l, W_r]$); say $B_1 \in [B_l, B_r]$ and $B_3 \in [W_l, W_r]$. If $B_2 \in [B_l, B_r]$, then $B_r \longrightarrow B_3$ and $B_3 \longrightarrow B_l$, which implies that B_l, B_r, $F_r(B_r)$ is a semiblock path. Similarly, if $B_2 \in [W_l, W_r]$, then W_l, W_r, $F_r(W_r)$ is a semiblock path.

Case 2: ϕ contains no semiblock B such that $F_r(B) \longrightarrow F_l(B)$. In this case, B_l, B_r, W_l, W_r is a semiblock path.

Note that $F_r(B_r) \xrightarrow{=} W_r$, thus B_l, B_r, $F_r(B_r)$ is a semiblock path if and only if $F_r^2(B_r) \neq W_r$. Similarly, W_l, W_r, $F_r(W_r)$ is a semiblock path if and only if $F_r^2(W_r) \neq B_r$. By Cases 1 and 2, we can compute a semiblock path in $O(1)$ time. $\square$

In the case of `reception(`$\tilde{B}_l$`, `$\tilde{B}_r$`)` we have to modify both $\mathcal{P}(\phi_1)$ and $\mathcal{P}(\phi_2)$ to obtain $\mathcal{P}(\psi)$, where ϕ_1 and ϕ_2 are the contigs referenced by $\tilde{B}_l$ and $\tilde{B}_r$, and ψ is the $\{v\}$-reception of $\langle B_l, B_r\rangle$. This update is applied after `reception` is completed, thus we have access to a semiblock pointer of $\{v\}$. The following lemma shows how to obtain $\mathcal{P}(\psi)$ spending no more time than the required for `reception`.

Lemma 4.2. *Let ϕ_1 and ϕ_2 be two (possibly equal) contigs such that $\{\phi_1, \phi_2\}$ contains a receptive pair $\langle B_l, B_r\rangle$ for $B_l \in \mathcal{B}(\phi_1)$ and $B_r \in \mathcal{B}(\phi_2)$, and ψ be the $\{v\}$-reception of $\langle B_l, B_r\rangle$ in $\{\phi_1, \phi_2\}$. Given a semiblock pointer to $B = \{v\}$ in ψ, it takes $O(\#[B_l, B_r])$ time to transform the semiblocks paths $\mathcal{P}(\phi_1)$ and $\mathcal{P}(\phi_2)$ into a semiblock path of ψ.*

Proof. Recall that, by definition, $N[B] = [B_l, B_r]$ and $\{\phi_1, \phi_2\} = \{\psi\} \setminus \{B\}$. Consider the following alternatives.

Alternative 1: B is an end semiblock of ψ. Suppose B is the left end semiblock as the other case is analogous. By definition, $\phi_1 = \phi_2$, $B_l = R(B)$ is the left end semiblock of ϕ_1, and $B_r = F_r(B)$. Traversing $[B_l, B_r]$ in ψ, we can check if $B \longrightarrow B_2$ for the semiblock B_2 that follows B_l in $\mathcal{P}(\phi_1)$. If affirmative, then a semiblock path of ψ is obtained by replacing B_l with B in $\mathcal{P}(\phi_1)$; otherwise, a semiblock path of ψ is obtained by inserting B before B_1 in $\mathcal{P}(\phi_1)$.

Alternative 2: B is not an end block of ψ, thus $B \in (B_l, B_r)$ in ψ, $B_l = F_l(B)$, and $B_r = F_r(B)$. Traversing $[B_l, B)$, we can check if $F_r(B_r) \in [B_l, B)$. If affirmative, then B is long and B, B_r, B_l is a semiblock path. Suppose, from now on, that $B_r \not\longrightarrow B_l$. By traversing $[B_l, B)$ we can find the leftmost semiblock B_a and the rightmost semiblock B_b of $\mathcal{P}(\phi_1)$ such that $B_a \longrightarrow B$ and $B_b \longrightarrow B$ (possibly $B_a = B_b$). Similarly, we can obtain the leftmost semiblock B_c and the rightmost semiblock B_d of $\mathcal{P}(\phi_2)$ such that $B \longrightarrow B_c$ and $B \longrightarrow B_d$. Note that these semiblocks exist because ψ is a contig. If $F_r^{\psi}(B_b) \neq B$, then B_b is not an end semiblock of ϕ_1, thus $\phi_1 = \phi_2$ and $B_b \longrightarrow B_c$; consequently, $\mathcal{P}(\phi_1)$ is a semiblock path of $\mathcal{P}(\psi)$. Otherwise, B_b is the right end semiblock of ϕ_1 and B_c is the leftmost end semiblock of ϕ_2 (perhaps $\phi_1 = \phi_2$). Then, the ordering obtained from $\mathcal{P}(\phi_1)$ by inserting B between B_a and B_d (removing B_b if $B_a \neq B_b$ and B_c if $B_c \neq B_d$) is a semiblock path of ψ.

Using the path pointers, we can apply all the modifications required on $\mathcal{P}(\phi_1)$ and $\mathcal{P}(\phi_2)$ in $O(1)$ time. We conclude, therefore, that $O(\#[B_l, B_r])$ time suffices to transform $\mathcal{P}(\phi_1)$ and $\mathcal{P}(\phi_2)$ into a semiblock path of ψ. □

Finally, to update $\mathcal{P}(\psi)$ after `remove(`$\tilde{B}$`)`, where ψ is the contig referenced by $\tilde{B}$, we have to revert the process done in the previous lemma. After the completion of `remove`, we obtain at most three contigs, namely ϕ_1, ϕ_2, and γ, where ϕ_1 is referenced by $L(\tilde{B})$, ϕ_2 is referenced by $R(\tilde{B})$, and γ is referenced by $\tilde{B}$. The computation of $\mathcal{P}(\gamma)$ is trivial; to compute $\mathcal{P}(\phi_1)$ and $\mathcal{P}(\phi_2)$ we apply the following lemma **before** invoking `remove`.

Lemma 4.3. *Let B be a semiblock of a contig ψ. Given a semiblock pointer to B, it takes $O(\#[F_l(B), F_r(B)])$ time to transform $\mathcal{P}(\psi)$ into $\{\mathcal{P}(\phi_1), \mathcal{P}(\phi_2)\}$, where $\{\phi_1, \phi_2\}$ is the family of contigs of $\{\psi\} \setminus \{B\}$.*

Proof. If $B \notin \mathcal{P}(\psi)$, then $\phi_1 = \phi_2$ and $\mathcal{P}(\psi)$ is a semiblock path of ϕ_1. Suppose, then, that $B \in \mathcal{P}(\psi)$ and consider the following alternatives.

Alternative 1: B is an end semiblock of ψ. Suppose B is a left end semiblock, as the other case is analogous, and note that $\phi_1 = \phi_2$ and $B_l = R(B)$ is the leftmost end semiblock of ϕ_1. Let $\mathcal{P}$ be the ordering obtained by replacing B with B_l in $\mathcal{P}(\phi_1)$. In $O(1)$ time we can obtain the first two semiblocks B_2 and B_3 that follow B in $\mathcal{P}(\phi_1)$. If $F_r(B_l) \neq B_3$, then $\mathcal{P}$ is a semiblock ordering of ϕ_1; otherwise, the ordering obtained by removing B_2 from $\mathcal{P}(\phi_1)$ is a semiblock ordering of ϕ_1.

Alternative 2: B is not an end semiblock of ψ, thus $B \in (B_l, B_r)$ for $B_l = F_l(B)$ and $B_r = F_r(B)$. First we search if ϕ_1 has a long semiblock. This happens only if $|\mathcal{P}(\psi)| = 3$, in which case $\phi_1 = \phi_2$, $B_r \longrightarrow B_l$, and, thus, $F_r^2(W) \in [B_l, W]$ for every $W \in (B, B_r]$. Marking the position of every semiblock in $[B_l, B_r]$, we can check in $O(1)$ time if $(F_r^{\phi_1})^2(W)$ appears after $F_l(W)$ in $[B_l, B_r]$ for some $W \in (B, B_r]$. If affirmative, then W is long and $F_l(W)$, W, $F_r(W)$ is a semiblock path of ϕ_1; otherwise, ϕ_1 has no long semiblocks. When ϕ_1 has no long semiblocks, we traverse $[B_l, B_r]$ to check if 1. $B_a \longrightarrow R(B)$, 2. $L(B) \longrightarrow B_b$, 3. $R(B) \longrightarrow B_{b+1}$, and $B_{a-1} \longrightarrow L(B)$, where B_{a-1}, B_a, B_b, and B_{b+1} are the semiblocks of $\mathcal{P}(\psi)$ such that $B_{a-1} \longrightarrow B_a$, $B_a \longrightarrow B$, $B \longrightarrow B_b$, and $B_b \longrightarrow B_{b+1}$ (unless $L(B_a) = \bot$ or $R(B_b) = \bot$ in which case $B_{a-1} = \bot$ and $B_{b+1} = \bot$, respectively). Replacing B with $R(B)$ if 1. and removing B_b if 3., or replacing B with $L(B)$ if 2. and removing B_a if 4., we obtain a semiblock path of $\phi_1 = \phi_2$. If neither 1. nor 2. holds, then we transform $\mathcal{P}(\psi)$ into the ordering $\mathcal{P}$ that is obtained by first replacing B with $L(B), R(B)$ and then removing B_b if 3. and B_a if 4. If $L(B) \longrightarrow R(B)$ or $\mathcal{P}(\psi)$ is circular, then $\mathcal{P}$ is a semiblock path of $\phi_1 = \phi_2$; otherwise, $\phi_1 \neq \phi_2$, thus we split $\mathcal{P}$ into the suborderings that have $L(B)$ as rightmost and $R(B)$ as leftmost to obtain semiblock paths of ϕ_1 and ϕ_2, respectively.

□

4.3 Round representations

To implement a round representation Φ we use a pair of doubly linked list $\{\tilde{\Phi}, \tilde{\Phi}^{-1}\}$ and a *connectivity* structure (see below). For each $\phi \in \Phi$, a semiblock pointer associated to ϕ (resp. ϕ^{-1}) is kept in $\tilde{\Phi}$ (resp. $\tilde{\Phi}^{-1}$). (The semiblock $\tilde{B}$ of ϕ plays the same role as the pointer to the first node in a linked lists when implementing the abstract data type; that is, $\tilde{B}$ is used to access ϕ.) Thus, both *physical* contigs ϕ and ϕ^{-1} are stored for each contig $\phi \in \Phi$. The reason why ϕ^{-1} is kept is to avoid the cost of reversing ϕ. If $\tilde{B} \in \tilde{\Phi} \cup \tilde{\Phi}^{-1}$ is associated to ϕ, then $B \in \mathcal{P}(\phi)$; moreover, B is the left end semiblock of ϕ when ϕ is linear. Conversely, B keeps a *contig pointer* to the position of $\tilde{B}$ inside $\tilde{\Phi} \cup \tilde{\Phi}^{-1}$. The contig pointer is used, among other things, to remove $\tilde{B}$ from $\tilde{\Phi} \cup \tilde{\Phi}^{-1}$ when its associated contig is joined to some other contig. Of course, this pointer has a null value when B is not referenced by a pointer in $\tilde{\Phi} \cup \tilde{\Phi}^{-1}$. Finally, each vertex v of a contig $\phi \in \Phi$ keeps a *reverse* pointer to its incarnation in ϕ^{-1}.

Recall that all the contigs of a round representation Φ are linear when $|\Phi| > 1$; this invariant must be satisfied by the data structure. Thus, we need some way to detect if an operation on a linear contig ϕ yields a circular contig ψ. Actually, the only operation in which we are ignorant about the linearity of ψ is when we compute the $\{v\}$-reception of $\langle B_l, B_r\rangle$. As it is shown in [13], the only possibility for ψ to be circular is when $[B_l, B_r)$ contains the right end semiblock of ϕ. To detect this case, we need to know if two end semiblocks belong to the same contig. As it was proved in [13], the connectivity problem is not solvable in $O(1)$ time when both insertions and removal of semiblocks are allowed. Thus, a *connectivity* data structure is kept to solve this problem. Its interface provides the following operations:

`create()` returns an empty connectivity structure. Complexity: $O(1)$ time.

`add(`$\tilde{B}$`)` adds $\tilde{B}$ to the connectivity structure. Complexity: $O(\text{ep}^+)$ time.

`remove(`$\tilde{B}$`)` removes $\tilde{B}$ from the connectivity structure. Complexity: $O(\text{ep}^-)$ time.

`opposite(`$\tilde{B}$`)` returns a pointer the other end semiblock of the contig that contains B if B is an end semiblock, or $\perp$ if B is not an end semiblock. Complexity: $O(\mathrm{ep}^+)$ time.

There are three flavors of the structure according to the operations supported by the main algorithm. In the *incremental* structure $O(\mathrm{ep}^+) = O(1)$ and $O(\mathrm{ep}^-) = O(n)$, in the *decremental* structure $O(\mathrm{ep}^+) = O(n)$ and $O(\mathrm{ep}^-) = O(1)$, and in the *fully dynamic* structure $O(\mathrm{ep}^+) = O(\mathrm{ep}^-) = O(\log n)$ [13, 30].

Let B_1 and B_2 be semiblocks of a round representation Φ that belong to the contigs ϕ_1 and ϕ_2 of Φ, respectively. To traverse the range $[B_1, B_2]$ of $\{\phi_1, \phi_2\}$, we have to provide a semiblock pointer $\tilde{B}_i$ to B_i associated to ϕ_i, for $i \in \{1, 2\}$. If, in turn, the semiblock pointer $\tilde{B}_1$ is associated to ϕ_1^{-1}, we obtain the range $[B_1, B_2]$ of $\{\phi_1^{-1}, \phi_2\}$. Thus, to describe the effects of an algorithm, we must specify the contig to which a pointer is associated. We say that $\tilde{B}$ has *type* Φ when $\tilde{B}$ is associated to some contig $\phi \in \Phi$. Those pointers of type Φ are sometimes referred to as Φ*-pointers*. Of course, every semiblock pointer has type either Φ of Φ^{-1}. Each semiblock pointer to B of type Φ^{-1} is called a *reverse* of $\tilde{B}$. We recall that there is no efficient way to know the type of semiblock pointer and, in general, the type is not important in the implementation. The purpose of this terminology is to aid in the specification of the different operations.

To avoid dealing with the pointers of round representations, we implement several operations that define an interface similar to the one used for contigs. As usual, we use capital Greek letters for round representations, capital Roman letters for semiblocks, and tildes for pointers.

`straight(`Φ`)` returns true if Φ is straight. For the implementation we test if a pointer in $\tilde{\Phi}$ references an end semiblock. Complexity: $O(1)$ time.

`newContig(`Φ`)` adds to Φ a new contig whose only block is $B = \{v\}$, and returns $\tilde{v}$. Requires Φ to be straight. For the implementation, we add a new physical contig to both $\tilde{\Phi}$ and $\tilde{\Phi}^{-1}$. Complexity: $O(1)$ time.

`reversed(`$\tilde{B}$`)` returns a reverse of $\tilde{B}$. For the implementation, we call `semiblock(`$\tilde{w}$`)` where $\tilde{w}$ is the reverse pointer of any $v \in B$. Complexity: $O(1)$ time.

`type(`$\tilde{B}$`)` returns the type of $\tilde{B}$, i.e., a pointer to either Φ or Φ^{-1}. Requires B to be an end semiblock. Let ϕ be the contig referenced by $\tilde{B}$. For the implementation we access the representation pointer of the physical semiblock referenced by either $\tilde{B}$ (if B is a left end semiblock in ϕ) or a reverse of $\tilde{B}$ (if B is a right end semiblock in ϕ^{-1}). Complexity: $O(1)$ time.

`reverse(`$\tilde{B}$`)` reverses the contig of Φ that contains B. Requires B to be an end semiblock. Applying `reversed` if required, assume B is a left end semiblock of a contig ϕ. Moreover, suppose, w.l.o.g., that $\tilde{B} \in \tilde{\Phi}$, as the other case is analogous. For the implementation, we use the contig and round pointers of B to move $\tilde{B}$ from $\tilde{\Phi}$ to $\tilde{\Phi}^{-1}$. Then, we use the connectivity structure together with `reversed` to obtain a Φ^{-1}-pointer $\tilde{W}$ to the right end semiblock of ϕ. Finally, we move $\tilde{W}$ from $\tilde{\Phi}^{-1}$ to $\tilde{\Phi}$. Complexity: $O(\mathrm{ep}^+)$ time.

`receptive(`$\tilde{B}_l$`, `$\tilde{B}_r$`)` takes two Φ-pointers and returns true if $\langle B_l, B_r \rangle$ is receptive in Φ. Let ϕ_1 and ϕ_2 be the contigs referenced by $\tilde{B}_l$ and $\tilde{B}_r$, respectively. For the implementation, observe that $\langle B_l, B_r \rangle$ is receptive if and only if $\langle B_l, B_r \rangle$ is receptive in $\{\phi_1, \phi_2\}$ and either the $\{v\}$-reception ψ of $\langle B_l, B_r \rangle$ is linear or $\phi_1 = \phi_2$ is the unique contig in Φ. Note that ψ is linear if and only if $[B_l, B_r)$ has no right end semiblocks or W and $\hat{R}(W)$ lie in different contigs, where $W \in [B_l, B_r)$ a right end semiblock. If $W \in [B_l, B_r)$ is an end semiblock, then we can check if ϕ_1 is the unique contig in Φ using the representation pointer of $\hat{R}(W)$.

To check if W and $\hat{R}(W)$ lie in different contigs, a query to the connectivity structure is required. Complexity $O(\#[B_l, B_r] + \mathrm{ep}^+)$ time.

`reception(`$\tilde{B}_l$`, `$\tilde{B}_r$`)` takes two Φ-pointers and updates Φ into the $\{v\}$-reception Ψ of $\langle B_l, B_r\rangle$. Requires $\langle B_l, B_r\rangle$ to be receptive in Φ, and returns a pointer to v. For the implementation, suppose ϕ_1 and ϕ_2 are the contigs referenced by $\tilde{B}_l$ and $\tilde{B}_r$, respectively. The first step is to apply `reception(`$\tilde{B}_l$`, `$\tilde{B}_r$`)` to transform ϕ_1 and ϕ_2 into a contig ψ that represents the v-reception of $\{\phi_1, \phi_2\}$. Let $W_l = R^{\psi}(\{v\})$, and observe that W_l is a left end block of Φ if and only $F_l(W_l) = \{v\}$. In this case, we access the contig pointer of W_l to remove it from $\tilde{\Phi}$. Similarly, if $\{v\}$ is a left end semiblock of Ψ, then we add a semiblock pointer to $\{v\}$ in Φ and we update the contig pointer of $\{v\}$. Finally, we test if $\tilde{\Phi} = \emptyset$. This happens when $\phi_1 = \phi_2$ is linear and ψ is circular, in which case $\{v\}$ belongs to $\mathcal{P}(\psi)$. Thus, again, we add a pointer to $\{v\}$ in $\tilde{\Phi}$. Once the update is completed, we apply the same procedure to `reversed(`$\tilde{B}_r$`)` and `reversed(`$\tilde{B}_l$`)` to update $\tilde{\Phi}^{-1}$. Finally, we update the connectivity structure. Complexity: $O(\#[B_l, B_r] + \mathrm{ep}^+)$ time.

`remove(`Ψ`, `$\tilde{B}$`)` transforms Ψ into a round representation Φ of $G(\Psi) \setminus B$. Requires $\tilde{B}$ to be of type $\Psi' \in \{\Psi, \Psi^{-1}\}$. For the implementation, we first call `remove(`$\tilde{B}$`)` to transform the contig $\psi \in \Psi'$ referenced by $\tilde{B}$ into two (possible equal) physical contigs ϕ_1 and ϕ_2, where ϕ_1 contains $\hat{L}(B)$. The next step depends on whether the contig pointer of B is null or not. In the latter case, we have to replace B in $\tilde{\Psi}'$ with a pointer to a semiblock W_l in $\mathcal{P}(\phi_2)$. Note that W_l must be a left end semiblock if ϕ_2 is linear; such a case occurs only when $W_l = R(B)$ is a left end semiblock of ϕ_2. In the former case, we check if $W_l = R^{\psi}(B)$ is a left end semiblock. If affirmative, then there are two possibilities. If $\Psi = \{\psi\}$ is not straight, then we replace the unique pointer in $\tilde{\Psi}$ with a pointer to W_l. If negative, then we insert W_l to $\tilde{\Psi}$. After completion, we apply the same transformations to $\tilde{\Psi}^{-1}$ using a reverse of $\tilde{B}$. Finally, we remove B from the connectivity structure. We remark that the obtained round representation Φ is not necessarily equal to $\Psi \setminus \{B\} = (\Psi \setminus \{\psi\}) \cup \{\phi_1, \phi_2\}$. The reason is that we are not aware of the type of $\tilde{B}$. Thus, if $\tilde{B}$ has type Ψ^{-1}, then when we decide to insert $R^{\psi}(B)$ to $\tilde{\Psi}$, we are actually computing $\Psi \setminus \{\psi\} \cup \{\phi_1, \phi_2^{-1}\}$. Complexity: $O(\#[F_l(B), F_r(B)] + \mathrm{ep}^-)$.

`separate(`$\tilde{B}$`, `W`)`**,** `separate(`W`, `$\tilde{B}$`)`**, and** `compact(`$\tilde{B}$`)` have the same effects as their contig versions on the contig of Φ that contains B. For the implementation, we apply the corresponding operations on ϕ and ϕ^{-1}, where ϕ is the contig referenced by $\tilde{B}$. Also, we take care of the contig and round pointers when B is an end semiblock. The details are similar to those described for the previous operations. Complexity: $O(|W|)$ time for `separate` and $O(\min\{|B|, |R(B)|\}$ time for `compact`.

`join(`$\tilde{B}_1$`, `$\tilde{B}_2$`)` takes a Φ_1-pointer B_1 and a Φ_2-pointer B_2 and builds the round representation $\Psi = \{\psi\}$ such that ψ satisfies the same specifications as the contig version of `join`. Requires B_1 and B_2 to be left co-end blocks and $\Phi_1 \neq \Phi_2$. The differences between this version of `join` and the one for contigs are the following. First, Φ_i could be a disconnected co-contig for $i \in \{1, 2\}$. In this case, Φ_i has only two contigs, each of which represents a clique. Second, the output Ψ is implemented as a round representation. To compute `join`, we apply one, two, or three calls to the contig version of `join`, according to whether Φ_1 and Φ_2 are disconnected or not. Since ψ is connected, it takes $O(1)$ time to restore all the pointers required by the data structure of Ψ. Finally, the connectivity structure can be updated in $O(\min\{\mathrm{ep}^+, \mathrm{ep}^-\})$ time as discussed in [30]. Time complexity: $O(u + \min\{\mathrm{ep}^+, \mathrm{ep}^-\})$ time, where u is the number of universal semiblocks in Ψ.

`split(`B_1`, `B_2`)` has the same effects as its contig version, but it returns the co-contigs implemented as round representations. Requires B_1 and B_2 to be left end blocks of the same type. Note that Ψ has a unique contig $\{\psi\}$, thus generating the pointer of the output representations is trivial. The connectivity structure can be updated in $O(\min\{\mathrm{ep}^+, \mathrm{ep}^-\})$ time as well [30]. Complexity: $O(u + \min\{\mathrm{ep}^+, \mathrm{ep}^-\})$ time, where u is the number of universal semiblocks in Ψ.

4.4 The witnesses

From the point of view of the end user, the dynamic algorithm keeps a round block representation Γ of the dynamic graph $G(\Gamma)$. To work with Γ, users can iterate through the semiblock pointers associated to contigs in Γ, while they execute the operations described in Section 4.1. However, only those operations that do not modify the internal structure of contigs are available, e.g., `vertices`, L, F_l, etc. To update $G(\Gamma)$, one of the following operations is applied.

`create()` returns an empty round block representation Γ. Complexity: $O(1)$ time.

`insert(`Γ`, `N`)` transforms Γ into a round block representation Ψ of a graph H such that $H \setminus \{v\} = G(\Gamma)$ for some vertex $v \notin V(\Gamma)$ with $N(v) = N$. Returns the new vertex v. The operation fails if H is not PCA and, in this case, a *minimally forbidden* of H is obtained (see below). Complexity: $O(|N| + \mathrm{ep}^+)$ time.

`remove(`v`)` transforms Γ into a round block representation Φ of $G(\Gamma) \setminus \{v\}$, where Γ is the round block representation containing v. Complexity: $O(d(v) + \mathrm{ep}^-)$ time.

Our goal is not only to implement the above operations that deal with PCA graphs, but also to provide a certifying algorithm for the recognition of PIG graphs. With respect to the positive witness, the latter problem is solved by satisfying the *straightness invariant* that guarantees that every contig of Γ is linear when $G(\Gamma)$ is a PIG graph. Regarding the negative certificate, we implement the following operation.

`forbiddenPIG(`Γ`)` returns a minimally forbidden witnessing that $G(\Gamma)$ is not PIG (i.e., a structure that represents a graph of Theorem 2.2). Complexity: $O(1)$ time.

When `insert(`Γ`, `$N(v)$`)` is executed and H is not PCA, for $H \setminus \{v\} = G(\Gamma)$, the end user obtains a negative witness. We say that a pair $\langle \Phi, \mathcal{N} \rangle$ is a *forbidden* of H (w.r.t. Γ) when:

- Φ is a round block representation of an induced subgraph of H,
- v is fully adjacent to every block in $\mathcal{N}$ and not adjacent to every block outside $\mathcal{N}$, and
- $H' = H[V(\Phi) \cup \{v\}]$ is not PCA.

If every subgraph of H' obtained by removing $B \in \mathcal{B}(\Phi)$ is PCA, then $\langle \Phi, \mathcal{N} \rangle$ is a *minimally* forbidden of H.

In case of failure, the output of `insert` is a minimally forbidden $\langle \Phi, \mathcal{N} \rangle$. To be useful to the end user, $\langle \Phi, \mathcal{N} \rangle$ has to be as efficient as possible. The least a user can expect is that L^Φ, R^Φ, F_l^Φ, and F_r^Φ take constant time. This allows the user to traverse the corresponding forbidden graph in $O(1)$ time per edge, and to take advantage of the PCA structure of $G(\Phi)$. Therefore, Φ is implemented with a data structure that satisfies these time bounds. As a consequence, finding a minimal $\mathcal{B} \subseteq \mathcal{B}(\Gamma)$ such that $H[\mathcal{B} \cup \{v\}]$ is not PCA is not enough. We also have to find the near and far neighbors of $\Gamma|\mathcal{B}$, and decide which vertices of $B \in \mathcal{B}$ survive when v is both adjacent and co-adjacent to B. As it is expected due to the time bounds, Φ shares some of the internal structure of Γ and, consequently, Φ must be discarded (or copied) before applying further operations on Γ.

5 An incremental and certified algorithm

This section is devoted to the implementation of `insert` (Section 4.4), whose aim is to insert a vertex $v \in V(H)$ into a round block representation Γ of $G = H \setminus \{v\}$ in $O(d(v) + \text{ep}^+)$ time. An algorithm for this problem was given in [30], and the method we present takes advantage of the tools developed in [30]. However, the algorithm in [30] is unable to output a minimally forbidden (or any witness whatsoever) when the insertion fails. The purpose of this section is to complete the algorithm by providing the negative witness. To show that our algorithm is correct, we prove of the Reception Theorem, following the same path as Deng et al. [8]. That is, we show a minimally forbidden of H when **no** round representation of G is v-receptive.

Because of the $O(d(v) + \text{ep}^+)$ time bound, we face two major inconveniences. First, we cannot traverse all the blocks of Γ. Thus, it is impossible to determine whether $B \longrightarrow W$ (in $O(d(v)+\text{ep}^+)$ time) when B and W are arbitrary blocks. This means that we need to infer some of the adjacencies by making appropriate queries on Γ. For this reason, in this section we assume that Γ, and every round representation obtained by transforming Γ, have been preprocessed as in the next observation, even if we are not explicit about this fact. This allows us to answer basic adjacencies queries as in Observation 5.2.

Observation 5.1 (see e.g. [30])**.** *Let H be a graph and Φ be a round representation of $H \setminus \{v\}$ for some $v \in V(H)$. Given $N(v)$ as input, it is possible to preprocess the semiblocks of Φ in $O(d(v))$ time so that determining whether v is (fully) adjacent to B can be answered in $O(1)$ time for any $B \in \mathcal{B}(\Phi)$ when a semiblock pointer to B is given.*

Observation 5.2. *Let H be a graph, Φ be a round representation of $H \setminus \{v\}$ for some $v \in V(H)$, and W_l, W_r be (possibly equal) semiblocks of Φ. Given semiblock Φ-pointers to W_l, W_r, the following problems can be solved in $O(\#[W_l, W_r])$ time:*

(a) obtain a Φ-pointer to the leftmost (resp. rightmost) semiblock of (W_l, W_r) co-adjacent to v.

(b) determine whether $W_l \longrightarrow W$ and $W \longrightarrow W_l$ (resp. $W \longrightarrow W_r$ and $W_r \longrightarrow W$), when a semiblock Φ-pointer to $W \in [W_r, F_r(W_r)]$ (resp. $W \in [F_l(W_l), W_l]$) is given.

The second inconvenience that arises when we want to compute a minimally forbidden, is that doing so requires a heavy amount of case by case analysis. The case by case analysis is somehow inherent to these kinds of proofs, as we need to proceed differently according to whether some edge exists or not (and the existence of such an edge may or may not imply the existence of other edges). To alleviate this situation, we make use of *adequate* forbidden. We say a family $\mathcal{B}$ of semiblocks of H is *forbidden* when $H[\mathcal{B}]$ is not PCA; $\mathcal{B}$ is *adequate* (with respect to a round representation Φ) when all the adjacencies between the semiblocks in $\mathcal{B}$ can be computed in $O(d(v))$ time when $N(v)$ and Φ are given as input. Clearly, if $\mathcal{B}$ is an adequate forbidden, then a minimally forbidden of H can be obtained in $O(d(v))$ time when $\mathcal{B}$, $N(v)$, and Φ are given. To prove that $\mathcal{B}$ is an adequate forbidden family we still have to prove that H' is not PCA. This task, however, can done by a computer (see Appendix A).

We divide our exposition in two major sections, according to whether G and H are co-connected or not. In the remaining of this section, we always use v to denote the vertex being inserted. So, for $B \in \mathcal{B}(\Phi)$ we write $+B$ and $-B$ as shortcuts for $B \cap N(v)$ and $B \setminus N(v)$, respectively, and we write $\pm B$ to mean a nonempty semiblock in $\{+B, -B\}$. Also, we write v as a shortcut for $\{v\}$ when a semiblock of H is expected.

5.1 Both H and G are co-connected

Throughout this section we consider that both H and $G = H \setminus \{v\}$ are co-connected. The advantage of this case is that $\Psi \cup \Psi^{-1} = \Gamma \cup \Gamma^{-1}$ for every pair of block co-contigs Ψ and Γ

representing G [16]. By (rec$_1$), we obtain that H is PCA only if the blocks fully adjacent to v are consecutive in some block co-contig representing G, say Γ. In this case, Γ (and Γ^{-1}) can be associated to at most two co-contigs that simultaneously satisfy (rec$_1$) and (rec$_2$). Therefore, it suffices to consider only these $O(1)$ co-contigs to prove the Reception Theorem. We first show how to obtain a minimally forbidden when $N(v)$ is consecutive in none of the block co-contigs representing G. But, before dealing with the consecutiveness of $N(v)$, we solve a rather restricted case in which the input representation Γ contains some "bad" blocks. The existence of such "bad" blocks is what makes it hard to test whether two blocks of Γ are adjacent. Without this hurdle, we can answer more powerful adjacencies queries.

We say that a semiblock B of a co-contig Φ is *good* when v is not adjacent to B or v is fully adjacent to all the semiblocks in either $[F_l(B), B)$ or $(B, F_r(B)]$. If B is good and v is either fully or not adjacent to B, then B is *perfect*, while B is *bad* when it is not good. It is not hard to see that Φ satisfies (rec$_1$) only if all its semiblocks are perfect. For such a co-contig Φ to exist, all the blocks of Γ must be good. Lemma 5.6 shows how to obtain a minimally forbidden when some block in Γ is bad. We consider two prior cases in Lemmas 5.4 and 5.5 whose common parts appear in the next lemma.

Lemma 5.3. *Let H be a graph with a vertex v, Γ be a block co-contig representing $H \setminus \{v\}$, and T_1, T_2, T_3 be blocks of Γ such that $T_1 \longrightarrow T_2$ and $T_2 \longrightarrow T_3$. Given semiblock Γ-pointers to T_1, T_2, and T_3, a minimally forbidden of H can be obtained in $O(d(v))$ time when either of the following conditions holds.*

(a) v is co-adjacent to T_1 and T_3 and adjacent to every block in (T_1, T_3), and $T_1 \not\longrightarrow T_3$.

(b) v is co-adjacent to T_3 and to $W \in [T_1, T_2]$, and $T_3 \longrightarrow T_1$

(c) v is co-adjacent to T_1 and T_3 and adjacent to every block in (T_1, T_3), $T_3 \longrightarrow U_r(T_2)$, $T_1 \longrightarrow T_3$, $U_l(T_2) \longrightarrow T_1$, and $U_r(T_2) \longrightarrow U_l(T_2)$ is obtainable in $O(d(v))$ time.

(d) v is co-adjacent to T_1 and T_3 and adjacent to every block in (T_1, T_3), $T_1 \longrightarrow T_3$, $F_r(T_1) = F_r(T_3)$, $U_l(T_1) \longrightarrow F_l(T_1)$, and $F_r(T_1) \longrightarrow U_l(T_1)$ is obtainable in $O(d(v))$ time.

Proof. We provide $O(d(v))$ time algorithms to find an adequate forbidden in each case.

(a) First, we query if $T_3 \longrightarrow T_1$ as in Observation 5.2 (b) with input $W_l = T_1$, $W_r = T_2$, $W = T_3$; if false, then $\{+T_2,\ v,\ -T_1,\ -T_3\}$ induces a $K_{1,3}$ of H. Suppose, then, that $T_3 \longrightarrow T_1$. Let $a \geq 1$ be the minimum such that either $U_l^{2a+1}(T_1) = U_l^{2a-1}(T_1)$ or $T_3 \not\longrightarrow U_l^{2a}(T_1)$. It is not hard to observe that such a value a always exists because the blocks not adjacent to W form the range $(F_r(W), F_l(W))$ for every $W \in \mathcal{B}(\Gamma)$. Moreover (see Figure 6 (a)):

   (i) $U_l(T_1) \in (T_2, T_3)$ and $U_l^{2i+1}(T_1) \in [U_l^{2i-1}(T_1), T_3)$ for every $2 \leq i < a$,

   (ii) $U_l^{2i}(T_1) \in (U_l^{2i-2}(T_1), T_2]$ for every $1 \leq i \leq a$.

   Thus, by marking every block in $[U_r(T_3), U_l(T_3)]$, the sequence $U_l(T_1), \ldots, U_l^{2a+1}(T_1)$ can be obtained in $O(|(T_1, T_3)|) = O(d(v))$ time.

   If $T_3 \not\longrightarrow U_l^{2a}(T_1)$, then, by (i) and (ii), it follows that $\{v,\ -T_1,\ +U_l(T_1),\ \ldots,\ +U_l^{2a}(T_1),\ -T_3\}$ induces a co-cycle in H, whose semiblocks are all adjacent to T_2. Consequently, a minimally forbidden can be obtained in $O(|(T_1, T_3)|) = O(d(v))$ time. Suppose, then, that $T_3 \longrightarrow U_l^{2a}(T_1)$, thus $U_l^{2a+1}(T_1) = U_l^{2a-1}(T_1)$ and $U_l^{2a}(T_1)$ are the right co-end block of Γ.

   By (i), the co-end blocks of Γ belong to (T_1, T_3). Since Γ is co-contig, we conclude that T_1 and T_3 belong to the same co-contig range, thus v is fully adjacent to all the blocks

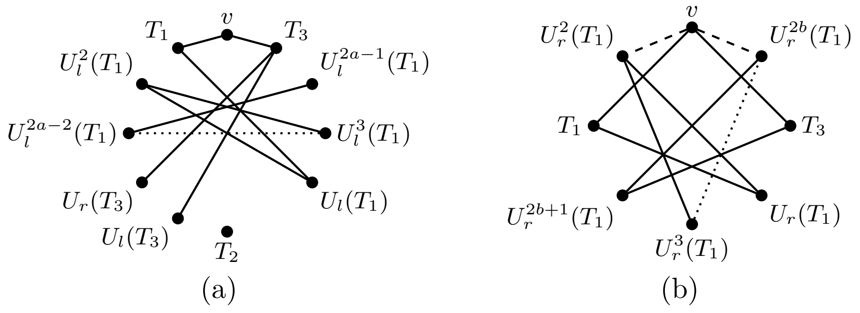


Figure 6: Adjacencies of $\overline{G(\Gamma)}$ in Lemma 5.3 (a). The blocks are drawn as they appear in the circular ordering $\mathcal{B}(\Gamma)$. Dotted lines are used for continuity, while dashed lines means an edge could be present or absent.

in the co-contig range that contains T_2. Let b be the minimum such that T_1, $U_r(T_1)$, $\ldots$, $U_r^{2b+1}(T_1)$, T_3 induces a co-path (see Figure 6 (b)). This co-path can be found in $O(d(v))$ time because v is fully adjacent to $U_r^{2i+1}(T_1)$ for every $0 \leq i \leq b$. Moreover, since $T_2 \longrightarrow T_3$, it follows that $b \geq 1$.

If $b > 1$ or v is fully adjacent to $U_r^2(T_1)$, then $\{v,\ -T_1,\ \pm U_r^1(T_1),\ \ldots,\ \pm U_r^{2b+1}(T_1),\ -T_3\}$ is an adequate forbidden (as it contains an induced $\overline{C_{2k}}$ ($k \geq 3$) or $\overline{H_2}$). If $b = 1$ and v is co-adjacent to $U_r^2(T_1)$, then, as $T_1 \longrightarrow T_2$ and $T_2 \longrightarrow T_3$, it follows that T_2, $U_r(T_1)$, T_3, $U_r^2(T_1)$, T_1, $U_r^3(T_1)$ appear in this order, thus T_2 is not adjacent to $U_r^2(T_1)$. Hence, $\{v,\ +T_2,\ +U_r(T_1),\ -T_3,\ -U_r^2(T_1),\ -T_1,\ +U_r^3(T_1)\}$ induces an $\overline{H_2}$ of H (A.1).

Before dealing with (b), we consider two subcases that are also solved in $O(d(v))$ time.

(e) v is co-adjacent to T_1, T_2, and T_3, and $T_3 \longrightarrow T_1$. By inspection, it can be observed that $\mathcal{B} = \{v,\ -T_1,\ -T_2,\ -T_3,\ \pm U_r(T_1),\ \pm U_r(T_2),\ \pm U_r(T_3)\}$ is a forbidden (A.2). Moreover, $\mathcal{B}$ is adequate because we can determine all the adjacencies in $O(1)$ time. Indeed, as $U_r(T_i) \in (T_{i+1}, T_{i-1})$, it follows that $U_r(T_i) \longrightarrow T_{i-1}$; since $U_r(T_i) \not\longrightarrow T_i$, then $U_r(T_i) \not\longrightarrow T_{i+1}$, while the adjacencies between $U_r(T_i)$ and $U_r(T_{i+1})$ are obtained in $O(1)$ time by Observation 5.2 (b) (with input $W_l = F_r(T_{i+1})$, $W_r = U_r(T_{i+1})$, and $W = U_r(T_i)$).

(f) v is co-adjacent to T_1 and T_3 and $T_3 \longrightarrow T_1$. Let W_l be the rightmost block in $[T_1, T_2]$ co-adjacent to v, and W_r be the leftmost block in $[T_2, T_3]$ co-adjacent to v. By Observation 5.2 (a), W_l and W_r can be obtained in $O(d(v))$ time. By Observation 5.2 (b), the query of whether $W_l \longrightarrow T_3$ can also be answered in $O(d(v))$ time. If affirmative, then we obtain a minimally forbidden by invoking (e) with input T_1, W_l, T_3. Otherwise, we obtain that $W_l \neq T_2$ and, thus, $W_r \neq T_2$. Again, by Observation 5.2 (b), $O(d(v))$ time suffices to find out whether $W_l \longrightarrow W_r$; if negative, then we obtain a minimally forbidden by invoking (a) with input W_l, T_2, W_r. When $W_l \longrightarrow W_r$, we query whether $T_3 \longrightarrow W_l$ using Observation 5.2 (b); if negative, then $\{+T_2,\ v,\ -W_l,\ -T_3\}$ induces a $K_{1,3}$, while if affirmative, then $W_r \neq T_3$ and we can obtain a minimally forbidden by invoking (e) with input W_l, W_r, T_3.

(b) Let W be the rightmost block in $[T_1, T_2]$ that is co-adjacent to v. If $W = T_1$, then we obtain a minimally forbidden by invoking (f). When $W \neq T_1$, we query whether $W \longrightarrow T_3$ and $T_3 \longrightarrow W$ using Observation 5.2 (b). If $W \longrightarrow T_3$, then we obtain a minimally forbidden by invoking (f) with input T_3, T_1, W. If $T_3 \longrightarrow W$, then we run (f) with input W, T_2, T_3. Finally, if $W \not\longrightarrow T_3$ and $T_3 \not\longrightarrow W$, then $\{+T_2,\ v,\ -W,\ -T_3\}$ induces a $K_{1,3}$.

(c) By Observation 5.2 (b), we can query in $O(d(v))$ whether $U_r(T_2) \longrightarrow T_1$ (resp. $T_3 \longrightarrow U_l(T_2)$); if so, then we obtain a forbidden using (b) with input T_1, T_3, $U_r(T_2)$ (resp. $U_l(T_2)$).

Otherwise, it follows that $U_l(T_2) \neq U_r(T_2)$ and, so, $\{v, -T_1, +T_2, -T_3, \pm U_l(T_2), \pm U_r(T_2)\}$ is a forbidden (A.3). Moreover, it is adequate as the only unknown edge is $U_r(T_2) \longrightarrow U_l(T_2)$ and this edge can be queried in $O(d(v))$ time.

(d) As Γ is a block representation, it cannot contain indistinguishable blocks. Thus, $F_l(T_2) \not\longrightarrow T_3$ and $F_l(T_1) \not\longrightarrow T_2$ (hence $F_l(T_1) \neq F_l(T_2)$ and $F_l(T_2) \neq T_1$). Observe that $F_r(T_1) \longrightarrow F_l(T_1)$ if and only if $F_r(T_1) \longrightarrow U_l(T_1)$ and $F_r^2(T_1) \neq U_l(T_1)$. By hypothesis, we can query, in $O(d(v))$ time, whether $F_r(T_1) \longrightarrow U_l(T_1)$; hence, we can also find out if $F_r(T_1) \longrightarrow F_l(T_1)$ in $O(d(v))$ time. Two cases then follow.

Case 1: $F_r(T_1) \longrightarrow F_l(T_1)$. We first check if v is co-adjacent to $F_r(T_1)$ or $F_l(T_1)$. If so, then we find a forbidden by calling (b) with input $T_1 = F_l(T_1)$, $T_2 = T_1$, and $T_3 = F_r(T_1)$ (in the former case) or $T_1 = T_1$, $T_2 = F_r(T_1)$, $T_3 = F_l(T_1)$ (in the latter case). Otherwise, $\mathcal{B} = \{v, -T_1, +T_2, -T_3, +U_l(T_1), +F_l(T_1), \pm F_l(T_2)\}$ is a forbidden (A.4).

Case 2: $F_r(T_1) \not\longrightarrow F_l(T_1)$. In this case, $\mathcal{B} = \{\pm U_l(T_1), \pm F_l(T_1), \pm F_l(T_2), -T_1, +T_2, -T_3, \pm F_r(T_1)\}$ is a forbidden (A.5).

Whichever the case, the forbidden $\mathcal{B}$ is adequate as it has at most two unknown edges, namely $F_r(T_1) \longrightarrow U_l(T_1)$ and $U_l(T_1) \longrightarrow F_l(T_2)$. The former can be queried in $O(d(v))$ time by hypothesis. To find out if the latter edge exists, we observe that $U_l(T_1) \longrightarrow F_l(T_2)$ if and only if $F_r(U_l(T_1)) \longrightarrow T_2$. Since $U_l(T_1) \longrightarrow F_l(T_1)$, then $F_r(U_l(T_1)) \in [F_l(T_1), T_1)$. So, by Observation 5.2 (b) —with input $W_l = T_1$, $W_r = T_2$, $W = F_r(U_l(T_1))$—, $O(d(v))$ time suffices to determine if $F_r(U_l(T_1)) \longrightarrow T_2$.

□

The next lemma describes how to find a forbidden when a long bad block B is given. Recall that B is long when $F_r(B) \longrightarrow F_l(B)$. How B was found, or why do we know that B is long are irrelevant questions at this point.

Lemma 5.4. *Let H be a graph with a vertex v, Γ be a block co-contig representing $H \setminus \{v\}$, and $B \in \mathcal{B}(\Gamma)$ be a long bad block. Given a semiblock Γ-pointer to B, a minimally forbidden of H can be obtained in $O(d(v))$ time.*

Proof. If v is co-adjacent to a block W in $[F_l^2(B), F_r(B)]$, then a minimally forbidden can be obtained by invoking Lemma 5.3 (b) with input $T_1 = F_l(B)$, $T_2 = B$, $T_3 = W$. Analogously, a minimally forbidden is obtained in $O(d(v))$ time when v is co-adjacent to a block in $[F_l(B), F_r^2(B)]$. Let W_l be the rightmost block in $(F_r^2(B), B)$ that is co-adjacent to v, and W_r be the leftmost block in $(B, F_l^2(B))$ that is co-adjacent to v. By Observation 5.2, W_l and W_r are found in $O(d(v))$ time, while we can also query whether $W_l \longrightarrow W_r$ in $O(d(v))$ time. If $W_l \not\longrightarrow W_r$, then we compute a minimally forbidden by invoking Lemma 5.3 (a). Using Observation 5.2 (a), we search for a block $W \in (F_r(B), F_l(B))$ co-adjacent to v. Moreover, when such a block exists, we query whether $W \longrightarrow W_l$ and $W_r \longrightarrow W$ with Observation 5.2 (b). When $W \longrightarrow W_l$ and $W_r \longrightarrow W$, we find a forbidden by calling Lemma 5.3 (b), while when $W \not\longrightarrow W_l$ (resp. $W_r \not\longrightarrow W$), the family $\{+F_l(B), v, -W_l, -W\}$ (resp. $\{+F_r(B), v, -W_r, -W\}$) induces a $K_{1,3}$. Therefore, we may suppose from now on that v is fully adjacent to every block in $[F_l^2(B), F_r^2(B)]$.

Note that, by hypothesis, $F_r(B) \longrightarrow U_l(B)$ and $U_r(B) \overset{=}{\longrightarrow} U_l(B)$. Hence, we can obtain a minimally forbidden in $O(d(v))$ by invoking Lemma 5.3 (c) with input $T_1 = W_l$, $T_2 = B$, and $T_3 = W_r$ when $F_l(W_l) \neq F_l(B)$ and $F_r(B) \neq F_r(W_l)$. Analogously, we obtain a minimally forbidden in $O(d(v))$ time when $F_r(W_l) = F_r(W_r)$, by calling Lemma 5.3 (d). By exchanging the roles of Γ and Γ^{-1} if required, suppose $F_r(B) = F_r(W_r)$ and, hence, $F_r(W_l) \neq F_r(B)$.

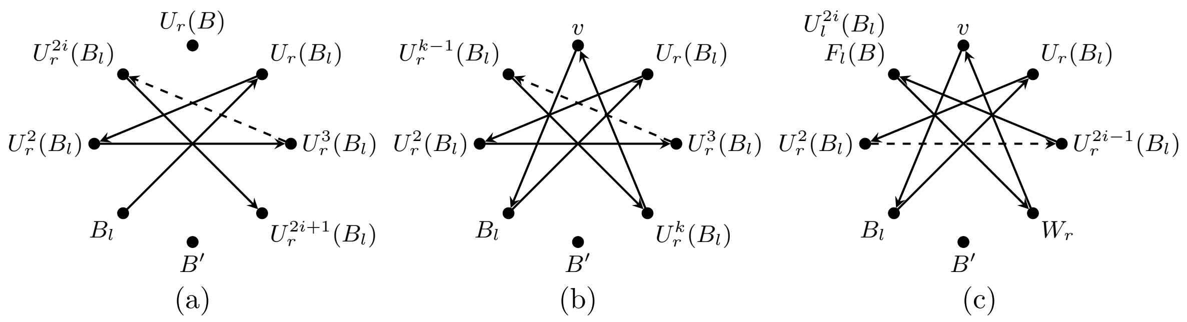


Figure 7: Co-paths of H in Lemma 5.4: (a) shows the position of the blocks in $\mathcal{B}(\Gamma)$, (b) depicts Case 1 for k odd, and (c) shows Cases 2 and 3.

Recall that $F_r(B) \not\to W_l$ by definition. For the next step, we proceed according to whether $U_r(B) = F_l(W_l)$ or not. If $U_r(B) \neq F_l(W_l)$, then $\{v, -W_l, +B, -W_r, +F_r(B), +U_r(B), +F_l(B)\}$ induces an $\overline{H_4}$ (A.6). When $U_r(B) = F_l(W_l)$, we determine in $O(1)$ time if $U_r(B)$ is a left co-end block by querying whether $U_r^3(B) = U_r(B)$. If $U_r(B)$ is not a left co-end block, then $B' = U_r^2(B) \in (W_l, B)$ is a block such that $F_r(B') \neq F_r(W_r) = F_r(B)$ and $F_l(B') \neq F_l(W_l) = U_r(B)$. In this case, we obtain a minimally forbidden by calling Lemma 5.3 (c) with input $T_1 = W_l$, $T_2 = B'$, and $T_3 = W_r$. (As $F_r(B') \longrightarrow U_r(B)$ and $F_l(B') \in (U_r(B), F_l(B))$, we can find out whether $F_r(B') \longrightarrow F_l(B')$ by traversing $(U_r(B), F_l(B))$. Recall that v is adjacent to every block in $(U_r(B), F_l(B))$, thus $O(d(v))$ time suffices to answer the query). We can suppose, then, that B' and $U_r(B)$ are left co-end blocks.

Let $B_l = U_r(F_r(B))$ if v is co-adjacent to $U_r(F_r(B))$, and $B_l = W_l$ otherwise. Since Γ is a co-contig, $U_r(B)$ and B' are the only left co-end blocks. Consequently $U_r^0(B_l), \ldots, U_r^k(B_l)$ induce a co-path for every $k > 0$ such that $U_r^k(B_l) \notin \{B', U_r(B)\}$. Moreover, the order of these blocks in $\mathcal{B}(\Gamma)$ is as in Figure 7 (a). Let k be the minimum such that either:

(i) $U_l^k(B_l)$ is co-adjacent to v,

(ii) k is odd and $U_r^k(B_l) \not\to F_l(B)$, or

(iii) k is even and $U_l^k(B_l) \not\to W_r$,

and consider the following cases.

Case 1: (i) holds; see Figure 7 (b). Then, $\mathcal{B} = \{v, -B_l, +U_r(B_l), \ldots, +U_r^{k-1}(B_l), -U_r^k(B_l)\}$ induces a co-cycle of length $k+2$. Suppose k is even and note that $U_r(B_l) \in [B', F_r(B)]$. Hence $U_r^2(B_l) \in [F_r(B), U_r(F_r(B))] \subseteq [F_r(B), B_l]$. Moreover, $U_r^2(B_l) \neq B_l$ because otherwise B_l would be a left co-end block different to B' and $U_r(B)$. Therefore, either $F_r(B) \longrightarrow U_r^2(B_l)$ or $U_r^2(B_l) = U_r(F_r(B)) \neq B_l$, which implies that v is fully adjacent to $U_r^2(B_l)$. Consequently, $k > 2$ and $\mathcal{B}$ is a forbidden. When k is odd, $B' \longrightarrow U_r^i(B_l)$ for every odd $1 \leq i \leq k$, while, because of the minimality of k, $U_r^i(B_l) \longrightarrow B'$ for every even $1 \leq i < k$. Therefore, $\mathcal{B} \cup \{B'\}$ is a forbidden.

Case 2: (ii) holds and (i) does not; see Figure 7 (c) with $k = i - 1$. Then, $\mathcal{B} = \{v, -B_l, +U_r(B_l), \ldots, +U_r^k(B_l)\}$ induces an odd co-path. Moreover, by the minimality of k, we obtain that $F_l(B)$ is adjacent to every block in $\mathcal{B} \setminus \{U_r^k(B_l)\}$, while B' and W_r are adjacent to every block in $\mathcal{B} \setminus \{v\}$. Hence, $\mathcal{B} \cup \{F_l(B), W_r, B'\}$ is a forbidden.

Case 3: (iii) holds and (i) does not; see Figure 7 (c) with $k = i$. This time, $\mathcal{B} = \{v, -B_l, -W_r, +U_r(B_l), \ldots, +U_r^k(B_l)\}$ induces an odd co-cycle. As in case (i), B' is adjacent to every block of $\mathcal{B} \setminus \{U_r^k(B_l)\}$, while, by the minimality of k, $U_r^k(B_l) \in [F_l(B), B]$, thus $U_r^k(B_l) \longrightarrow B'$. Consequently, $\mathcal{B} \cup \{B'\}$ is a forbidden.

To compute $U_r(B_l), \ldots, U_r^k(B_l)$, we proceed as follows. First, we mark with 1 the blocks in $[U_r(B), F_l(B))$ and with 2 the blocks in $[B, W_r]$. Then we traverse $U_r^k(B_l)$ for each $k > 0$ until we find a block co-adjacent to v or a marked block. In the former case (i) holds for k, while in the latter case (ii) or (iii) holds for $k-1$ when $U_r^k(B)$ is marked with 1 or 2, respectively. We conclude that $O(d(v))$ time is enough to find the forbidden. □

The case in which B is a short bad block is solved next. Note that, a priori, Γ could contain long bad blocks; yet, we have no evidence about the existence or non-existence of long bad blocks.

Lemma 5.5. *Let H be a graph with a vertex v, Γ be a block co-contig representing $H \setminus \{v\}$, and $B \in \mathcal{B}(\Gamma)$ be a short bad block. Given a semiblock Γ-pointer to B, a minimally forbidden of H can be obtained in $O(d(v))$ time.*

Proof. Let W_l be the rightmost block in $[F_l(B), B)$ co-adjacent to v, and W_r be the leftmost block in $(B, F_r(B)]$ co-adjacent to B. We can find W_l and W_r and test whether $W_l \longrightarrow W_r$ with Observation 5.2. If $W_l \not\longrightarrow W_r$, we find a minimally forbidden as in Lemma 5.3 (a).

By hypothesis, $F_r(B) \not\longrightarrow F_l(B)$; hence, $F_r(B) \longrightarrow U_l(B)$ if and only if $F_r^2(B) = U_l(B)$. Similarly, $U_r(B) \longrightarrow U_l(B)$ if and only if $U_r(B) \neq U_l(B)$ and either $F_r(B) \longrightarrow U_l(B)$ or $F_l(U_l(B)) = U_r(B)$. We conclude, therefore, that $O(1)$ time suffices to determine whether $F_r(B) \longrightarrow U_l(B)$ and whether $U_r(B) \longrightarrow U_l(B)$. Therefore, by Lemma 5.3 (c), we can suppose that $F_r(B) = F_r(W_r)$. Otherwise, we obtain a minimally forbidden in $O(d(v))$ time. Similarly, by Lemma 5.3 (d) we find a minimally forbidden if $F_r(W_l) = F_r(B)$ and $F_l(W_l)$ is not an end block. Then, two cases remain:

Case 1: $F_r(W_l) \neq F_r(B)$. Thus, $F_r(B) \neq W_r$ and, as Γ has no pair of indistinguishable blocks, $F_l(B) \not\longrightarrow W_r$ and, so, $F_l(B) \neq W_l$. Under this situation, $\{v, -W_l, +B, -W_r, \pm F_l(B), \pm F_r(B)\}$ is an adequate forbidden (A.7).

Case 2: $F_r(W_l) = F_r(B)$ and $F_l(W_l)$ is an end block. Note that $F_l(W_l) \neq F_l(B)$ and $F_l(B) \neq W_l$ because Γ has no indistinguishable blocks. If $F_r(W_l)$ is not a right end block, then let $X = U_r(B_l)$ and note that $W_r \not\longrightarrow X$. Otherwise, since W_l is not universal, it follows that Γ has some other contig with a block X. Whichever the case, $\{v, \pm F_l(W_l), \pm F_l(B), -W_l, +B, -W_r, \pm X\}$ is an adequate forbidden (A.8). We remark that X can be obtained in $O(1)$ time by using the representation and contig pointers of $F_l(W_l)$.

□

We are now ready to deal with the existence of bad blocks, regardless of their type. The main idea is to find a bad block B which can be used as input of either Lemma 5.4 or Lemma 5.5. In order to apply either lemma, we need to find out whether B is short or long. However, we do not know how to test, in $O(d(v))$ time, whether $F_r(B) \longrightarrow F_l(B)$ when only B is given. The solution to this problem is to take advantage of the dynamic nature of the algorithm. That is, the answer to $F_r(B) \longrightarrow F_l(B)$ was found when the last vertex of G was inserted and it is implicitly encoded in the semiblock paths.

Lemma 5.6. *Let H be a graph with a vertex v, and Γ be block co-contig representing $H \setminus \{v\}$. Given $N(v)$, it costs $O(d(v))$ time to test if Γ has bad blocks. Furthermore, if Γ has bad blocks, then a minimally forbidden of H can be obtained within the same amount of time.*

Proof. The algorithm has two main phases. In the first phase, all the bad blocks of Γ are marked; in the second phase a minimally forbidden is obtained.

To find the bad blocks we first mark all the blocks of Γ that are fully adjacent to v in such a way that B, W have the same mark if and only if B and W lie in the same contig, v is fully adjacent to all the blocks in $[B, W]$ (or $[W, B]$), and $[B, W)$ (or $[W, B)$) has no right end blocks. Then, a block B adjacent to v is bad if and only if B is not an end block and either:

- $L(B)$ and $R(B)$ are unmarked, or
- $R(B)$ and $F_r(B)$ have different marks, and $L(B)$ and $F_l(B)$ have different marks.

It is not hard to both steps can be achieved in $O(d(v))$ time (see e.g. [30]). After these steps we can test if any block in Γ is bad in $O(1)$ time.

Let $T_1, \dots, T_k$ be the blocks of $\mathcal{P}(\gamma)$ for any $\gamma \in \Gamma$. Recall that, by definition, Γ has a long block if and only if $k = 3$ and $\mathcal{P}(\gamma)$ is circular. Hence, if $k \neq 3$ or $\mathcal{P}(\gamma)$ is linear, then we obtain a minimally forbidden by invoking Lemma 5.5 with and any bad block as input. When $k = 3$ and $T_3 \longrightarrow T_1$, we first check if v is co-adjacent to T_1, T_2, and T_3; if true, then we obtain the minimally forbidden via Lemma 5.3 (b) with input T_1, T_2, T_3. Suppose, then, that v is fully adjacent to T_2. If T_2 is a bad block, then we obtain the forbidden with Lemma 5.4 using T_2 as input. Otherwise, v is fully adjacent to all the blocks in either $[T_2, F_r(T_2)]$ or $[F_l(T_2), T_2]$. Assume the former, as the proof for the latter is analogous.

Let W_r and W_l be the leftmost and rightmost blocks co-adjacent to v in $(F_r(T_2), T_2)$, respectively, and observe that $W_l \not\longrightarrow W_r$ because $T_2 \not\longrightarrow W_r$. We can both find W_l and W_r and query if $W_r \longrightarrow W_l$ in $O(d(v))$ time by Observation 5.2. If any $B \in (W_l, W_r)$ is bad, then $W_l \longrightarrow B$ and $B \longrightarrow W_r$ because v is fully adjacent to all the blocks in (W_l, W_r). When $W_r \not\longrightarrow W_l$, the family $\{v, +B, -W_l, -W_r\}$ induces a $K_{1,3}$, while when $W_r \longrightarrow W_l$ we find a minimally forbidden by calling Lemma 5.4 with input B. Whichever the case, $O(d(v))$ time is enough to obtain a minimally forbidden when some block in (W_l, W_r) is bad.

For the final case, suppose no block in (W_l, W_r) is bad, thus the bad block B belongs to (W_r, W_l). Note that either $F_r(B) \in (W_l, W_r)$ or $F_l(B) \in (W_l, W_r)$, we assume the former as the other case is analogous. This means that $F_r(B)$ is good, thus $F_r^2(B)$ belongs to (W_l, W_r) as well. Hence, we can check if $F_r^2(B) \longrightarrow B$ as in Observation 5.2 (b) (with input $W_l = F_r(B)$, $W_r = F_r^2(B)$, and $W = B$). Then we find the minimally forbidden calling Lemma 5.4 (if affirmative) or Lemma 5.5 (if negative) with input B. We conclude, therefore, that a minimally forbidden can be obtained in $O(d(v))$ time. $\square$

Having dealt with bad blocks, we now consider the case in which $N(v)$ is not consecutive in Γ. That is, we discuss how to find a forbidden when no co-contig representing G satisfies (rec_1). The core of the proof is given in the next lemma.

Lemma 5.7. *Let H be a graph with a vertex v, and Γ be a block co-contig representing $H \setminus \{v\}$ with no bad blocks. If $B_l \neq B_r$, and X are blocks of Γ such that:*

- *$X \notin [B_l, B_r] \cup \{L(B_l), R(B_r)\}$, and v is adjacent to every block in $[B_l, B_r] \cup \{X\}$,*
- *if $L(B_l) \neq \bot$, then v is co-adjacent $L(B_l)$; otherwise v is co-adjacent to B_l, and*
- *if $R(B_r) \neq \bot$, then v is co-adjacent $R(B_r)$; otherwise v is co-adjacent to B_r,*

then a (minimally) forbidden of H can be obtained in $O(d(v))$ time when semiblock Γ-pointers to B_l and B_r and a semiblock pointer $\tilde{X}$ to X are given.

Proof. The first step of the algorithm is to decide if $X \longrightarrow L(B_l)$. Even though we are unaware of the type of $\tilde{X}$, we can answer this query in $O(d(v))$ time by observing that, since B_l is good and v is co-adjacent to B_r or $R(B_r)$, then either $L(B_l) = \bot$ or $F_r(B_l) \in [B_l, B_r]$. In the former case $X \not\longrightarrow L(B_l)$. In the latter case $X \longrightarrow L(B_l)$ if and only if $F_r(X) \in [L(B_l), B_r]$, which happens if and only if $\tilde{X} \in [L(B_l), B_r]$ or $\tilde{X}^{-1} \in [L(B_l), B_r]$. (Here $\tilde{X}^{-1}$ is a reverse of $\tilde{X}$.) In a similar way, we can test if $R(B_r) \longrightarrow X$ in $O(d(v))$ time. If $X \not\longrightarrow L(B_l)$ and $R(B_r) \not\longrightarrow X$, then $R(B_r) \not\longrightarrow L(B_l)$ and:

- $\{v, +B_l, +B_r, +X\}$ induces a $K_{1,3}$ if $B_l \not\longrightarrow B_r$, and
- $\{v, +B_l, +B_r, +X, -L(B_l), -R(B_r)\}$ induces an $\overline{S_3}$ if $B_l \longrightarrow B_r$ (A.9). We remark that B_l is not the left end block in this case, thus $L(B_l) \neq \bot$. Otherwise, B_r is not the right end block and, since B_r is good and v is co-adjacent to B_l, it follows that v is fully adjacent to $(B_r, F_r(B_r)]$. This is a contradiction because v is co-adjacent to $R(B_r)$. In a similar way $R(B_r) \neq \bot$.

From now on suppose $R(B_r) \longrightarrow X$, as the proof when $L(B_l) \longrightarrow X$ is analogous. Hence, X and B_r lie in the same contig. Moreover, by applying `reversed` if required, we may assume that $\tilde{X}$ is a Γ-pointer, thus we can invoke Observation 5.2 whenever it is required.

Let $X_r = F_r(R(B_r))$, $W_l = B_l$ if $L(B_l) = \bot$, and $W_l = L(B_l)$ otherwise. Since B_r is good and $B_r \longrightarrow R(B_r)$, we observe that $W_l \not\longrightarrow B_r$. Similarly, since $R(B_r) \longrightarrow X$ and X is good, we obtain that v is adjacent to X_r and, since X_r is good, it follows that $X_r \not\longrightarrow W_l$. By checking if $F_l(X_r) \neq R(B_r)$, we can decide if $B_r \longrightarrow X_r$; if negative, then $\{v, +B_r, -R(B_r), +X_r, -W_l\}$ induces a C_4^*. Suppose, then, that $B_r \longrightarrow X_r$ (hence $B_r \longrightarrow X$), thus $F_l(B_r) \not\longrightarrow R(B_r)$ because Γ has no indistinguishable blocks. Next we query if $F_r(X) = X_r$. If affirmative, then $U_l(X) \longrightarrow R(B_r)$ because Γ has no indistinguishable semiblocks. So, $U_l(X) \in (F_l(B_l), B_r)$ is adjacent to v, which implies that $W_l \not\longrightarrow U_l(X)$. Consequently, $\{v, +U_l(X), -R(B_r), +X, -W_l\}$ induces a C_4^*. Finally, if $F_r(X) \neq X_r$, then $\{v, -W_l, +F_l(B_r), +B_r, -R(B_r), +X, +F_r(X)\}$ is a forbidden (A.10) which is adequate by Observation 5.2. □

We are now ready to find a minimally forbidden when $N(v)$ is not consecutive in Γ. Before doing so, it is convenient to state precisely what we mean by consecutive. We remark that the definition holds for any round representation and not only for co-contigs. We say that $N(v)$ is *consecutive* in a round representation Φ when there exist two (possibly equal) semiblocks B_a and B_b such that $N(v) \subseteq \bigcup[B_a, B_b] \cup +B_a \cup +B_b$. In such a case, $\langle B_a, B_b\rangle$ *witnesses* that $N(v)$ is consecutive in Φ. Clearly, if $\langle B_l, B_r\rangle$ satisfies (rec$_1$), then $N(v) = \bigcup[B_l, B_r]$ is consecutive in Φ. However, consecutiveness is a slight generalization of condition (rec$_1$) that allows v to be co-adjacent to B_a and B_b. The next result applies Lemma 5.7 to find a minimally forbidden when $N(v)$ is not consecutive in any co-contig representing $G(\Gamma)$.

Lemma 5.8. *Let H be a graph with a non-isolated vertex $v \in V(H)$, and Γ be block co-contig representing $H \setminus \{v\}$ with no bad blocks. Given $N(v)$, it takes $O(d(v) + \mathrm{ep}^+)$ time to transform Γ into a block co-contig Γ' representing $G(\Gamma)$ in which $N(v)$ is consecutive. The algorithm outputs Γ'-pointers to the blocks $\langle B_a, B_b\rangle$ witnessing that $N(v)$ is consecutive in Γ', or a minimally forbidden of H when such a representation Γ' does not exist.*

Proof. As discussed in [13, 30], $O(d(v))$ time suffices to find a set $\{\tilde{B}_i, \tilde{W}_i \mid 1 \leq i \leq k\}$ of semiblock pointers such that:

(a) $\tilde{B}_i$ and $\tilde{W}_i$ are associated to the same contig γ_i,

(b) v is fully adjacent to every block in (B_i, W_i),

(c) $[B_1, W_1], \ldots, [B_k, W_k]$ is a partition of the blocks adjacent to v, and

(d) either $L_i(B_i) = \bot$ (resp. $R_i(W_i) = \bot$) or v is co-adjacent to $L_i(B_i)$ (resp. $R_i(W_i)$).

We remark that the type of $\tilde{B}_i$ is unknown, as we are unaware of whether $\gamma_i \in \Gamma$ or $\gamma_i \in \Gamma^{-1}$. In (b) and (c) above, $[B_1, W_1]$ refers to the range in γ_i, that could be a range of Γ^{-1}. For the sake of notation, we write L_i and R_i as shortcuts for L^{γ_i} and R^{γ_i}, as in (d) above.

When $k = 1$, $\langle B_1, W_1\rangle$ witnesses that $N(v)$ is consecutive in Γ', where Γ' is the type of $\tilde{B}_1$. Suppose, then, that $k \geq 2$. If either none of $\{B_i, W_i\}$ $(1 \leq i \leq k)$ are end blocks or $B_i \neq W_i$ and v is co-adjacent to the end blocks in $\{B_i, W_i\}$, then we obtain a minimally forbidden by invoking Lemma 5.7 with input $B_l = B_i$, $B_r = W_i$, and $X = B_j$ for $j \neq i$. If $k \geq 3$, then $\{v, +E_1, +E_2, +E_3\}$ induces a $K_{1,3}$, where E_i is the end block of $\{B_i, W_i\}$. Therefore, $k = 2$, $E_i \in \{B_i, W_i\}$ $(i \in \{1, 2\})$ is an end block, and v is fully adjacent to E_i if $B_i \neq W_i$. Exchanging the roles of γ_i and γ_i^{-1} if required, we may assume that $E_1 = W_1$ is a right end block and $E_2 = B_2$ is a left end block. If B_2 has the same type Γ' as W_1, then $\langle B_1, W_2\rangle$ witnesses that $N(v)$ is consecutive in Γ'; otherwise, $\langle B_1, W_2\rangle$ witnesses that $N(v)$ is consecutive in $(\Gamma' \setminus \{\gamma_2\}) \cup \{\gamma_2^{-1}\}$. As discussed in Section 4.3, $O(1)$ time is enough to query the types of W_1 and B_2, while replacing γ_2 with γ_2^{-1} costs $O(\mathrm{ep}^+)$ time. □

By definition, $N(v) \neq \emptyset$ is consecutive in a round block representation Γ when it has two (possibly equal) blocks B_a and B_b such that $N(v) = \bigcup(B_a, B_b) \cup +B_a \cup +B_b$. We can separate B_a and B_b in pairs of consecutive indistinguishable semiblocks $\langle -B_a, +B_a\rangle$ and $\langle +B_b, -B_b\rangle$, respectively, to obtain a new round representation of G. Of course, if either $+B_a$ (resp. $-B_b$) or $-B_a$ (resp. $+B_b$) is empty, then nothing is separated out of B_a (resp. B_b). Similarly, if $B_a = B_b$, then $+B_a$ is separated to either the left or the right of $-B_a$. We refer to the round representation so obtained as being *v-associated* to Γ. Observe that v is simultaneously adjacent and co-adjacent to at most two blocks of Γ, namely B_a and B_b. Thus, no matter which pair witnesses that $N(v)$ is consecutive in Γ, only B_a and B_b could be separated. Moreover, both B_a and B_b get separated only if v is co-adjacent to both B_a and B_b, in which case either $B_a = B_b$ or $\langle B_a, B_b\rangle$ is the only pair witnessing that $N(v)$ is consecutive. We conclude, therefore, that at most two round representations v-associated to Γ exist.

Recall that, when v is not isolated, the co-connected graph H is PCA if and only if $G = H \setminus \{v\}$ admits a v-receptive round representation Φ. By the Reception Theorem (applied to the component that contains v), Φ has a pair of semiblocks $\langle B_l, B_r\rangle$ that satisfies (rec_1)–(rec_3). Recall that $\langle B_l, B_r\rangle$ satisfies (rec_1) if and only if $N(v) = \bigcup[B_l, B_r]$, while $\langle B_l, B_r\rangle$ satisfies (rec_2) when no pair of semiblocks in $\Phi \setminus \{B_l, B_r\}$ are indistinguishable. It is not hard to see that Φ satisfies (rec_1) and (rec_2) if and only if Φ is a round representation v-associated to Γ, for some round block representation Γ of G. Indeed, by (rec_2), Φ has at most two pair of indistinguishable semiblocks, namely $\{L(B_l), B_l\}$ and $\{B_r, R(B_r)\}$. By compacting $L(B_l) \cup B_l$ into B_a and $B_r \cup R(B_r)$ into B_b, we obtain a round block representation Γ that has Φ as its v-associated representation. We record this fact for future reference.

Observation 5.9. *Let H be a co-connected graph with a vertex non-isolated vertex v. A round representation Φ of $H \setminus \{v\}$ satisfies (rec_1) and (rec_2) if and only if Φ is v-associated to a round block representation of $H \setminus \{v\}$.*

By definition, a round representations is just a family of contigs with no order. Consequently, G admits only two round block representations in which $N(v)$ is consecutive, namely Γ and Γ^{-1}. Therefore, the only round representations of G that satisfy (rec_1) and (rec_2) are those v-associated with Γ and Γ^{-1}. We show how to obtain a minimally forbidden when none of the representations v-associated to Γ satisfies (rec_3). Before doing so, we find it convenient to recall that $\langle B_l, B_r\rangle$ satisfies (rec$_3$) when there exists $B_m \in [B_l, B_r]$ witnessing that $\langle B_l, B_r\rangle$ is receptive. Also, recall that B_m witnesses that $\langle B_l, B_r\rangle$ is receptive when (see Figure 4):

(wit$_\mathrm{a}$) B_m is an end semiblock, $B_l \xrightarrow{=} B_m$, and $B_m \xrightarrow{=} B_r$, or

(wit$_\mathrm{b}$) $B_l \xrightarrow{=} B_m$, $B_m \not\longrightarrow R(B_r)$, $L(B_l) \not\longrightarrow R(B_m)$, and $\hat{R}(B_m) \xrightarrow{=} B_r$.

Lemma 5.10. *Let H be a graph with a non-universal vertex $v \in V(H)$, Γ be a block co-contig representing $H \setminus \{v\}$ with no bad blocks, and $\langle B_a, B_b\rangle$ be a pair witnessing that $N(v) \neq \emptyset$ is consecutive in Γ. Given semiblock Γ-pointers to B_a and B_b, it takes $O(d(v))$ time to transform Γ into a co-contig Φ representing $H \setminus \{v\}$ that satisfies* (rec$_1$)*–*(rec$_3$)*. The algorithm outputs the semiblock Φ-pointers to a pair $\langle B_l, B_r\rangle$ that satisfies* (rec$_1$)*–*(rec$_3$)*, or a minimally forbidden of H when such a representation Φ does not exist.*

Proof. If $B_a = B_b$, then B_a is an end block because it is good. Thus, one of the representations Φ v-associated to Γ has $+B_a$ as an end semiblock and, so, $+B_a$ satisfies (wit$_\mathrm{a}$). Then, by Observation 5.9, $\langle +B_a, +B_a\rangle$ satisfies (rec$_1$)–(rec$_3$) in Φ. Suppose, from now on, that $B_a \neq B_b$. Let Φ be the co-contig v-associated to Γ such that $N(v) = \bigcup[B_l, B_r]$ for $B_l = +B_a$ and $B_r = +B_b$. By Observation 5.9, Φ satisfies both (rec$_1$) and (rec$_2$). With the operations discussed in Section 4.3, we can transform Γ into Φ and test if Φ satisfies (rec$_3$) in $O(d(v))$ time. If affirmative, then we return $\langle B_l, B_r\rangle$. Suppose, then, that Φ does not satisfy (rec$_3$).

Claim 1: $B_l \not\longrightarrow B_r$. If $B_l \longrightarrow B_r$, then neither B_l nor B_r is an end semiblock because $B_m \in \{B_l, B_r\}$ does not satisfy (wit$_\mathrm{a}$). So, $L(B_l) \longrightarrow B_l$ and $B_r \longrightarrow R(B_r)$ and, since B_l and B_r are good, it follows that $B_l = F_l(B_r)$ and $B_r = F_r(B_l)$. Consequently, $B_m = U_l(R(B_r)) \in [B_l, B_r)$, thus $B_l \xrightarrow{=} B_m$ and $R(B_m) \longrightarrow B_r$. Moreover, since $B_m \not\longrightarrow R(B_r)$ and B_m does not satisfy (wit$_\mathrm{b}$), it follows that $L(B_l) \longrightarrow R(B_m) = F_l(R(B_r))$, a contradiction because $R(B_m)$ is good.

Claim 2: $F_r(B_l) \not\longrightarrow B_r$. Suppose $F_r(B_l) \longrightarrow B_r$, thus $B_l \neq F_r(B_l)$ and $F_r(B_l) \neq B_r$. If $F_r^2(B_l) = B_r$, then $F_r(B_m)$ witnesses that $\langle B_l, B_r\rangle$ is receptive by (wit$_\mathrm{b}$). Otherwise, $F_r(B_l) \longrightarrow R(B_r)$, thus $B_m = U_l(R(B_r)) \in [B_l, B_r)$. Consequently, $B_l \xrightarrow{=} B_m$ and, since $B_m \not\longrightarrow R(B_r)$ and $R(B_m) \xrightarrow{=} B_r$ by definition, we obtain that $L(B_l) \longrightarrow R(B_m)$. This is a contradiction, because $R(B_m)$ is good and $R(B_m) \longrightarrow R(B_r)$.

Claim 3: $\hat{U}_r(B_l) \not\longrightarrow B_r$. Otherwise, $B_m = F_r(B_l)$ satisfies (wit$_\mathrm{b}$). Just note that, by Claim 2, $B_m \not\longrightarrow B_r$.

Let $B_m = \hat{U}_r(B_l)$ and $B'_r = \hat{U}_r(B_m)$. By Claims 1–3, we observe that B_l, $F_r(B_l)$, B_m, $F_r(B_m)$, B'_r, and B_r appear in this order in a traversal of $[B_l, B_r]$ where, possibly, $B_l = F_r(B_l)$, $B_m = F_r(B_m)$, and $B'_r = B_r$. In $O(d(v))$ time we can check whether $B'_r \longrightarrow B_l$; if negative, then $\{v, B_l, B_m, B'_r\}$ induces a $K_{1,3}$. Thus, we assume that $B'_r \longrightarrow B_l$. Hence, B'_r is not an end block and, by Claim 3, $B'_r, R(B_r), B_l$ are pairwise different and appear in this order in a traversal of the contig that contains B_l and B'_r.

Claim 4: if (B_r, B_l) has some semiblock W that is indistinguishable to neither B_l nor B'_r, then a minimally forbidden can be obtained in $O(d(v))$ time. By (rec$_2$), there are $O(1)$ blocks that are indistinguishable to either B_l or B'_r, thus W can be obtained in $O(1)$ time. Clearly, v is not adjacent to W by (rec$_1$). First we test if $B'_r \longrightarrow F_r(B_l)$ by looking whether $F_r(B'_r) = F_r(B_l)$. If affirmative, then, since no pair of B'_r, W, and B_l are indistinguishable, it follows that $W_a = U_l(W) \longrightarrow B'_r$ and $W_b = U_l(B_l) \longrightarrow W$. Note that, consequently, B_m, W_a, W_b, and B'_r are pairwise different. Hence, $\{v, +B_l, +B_m,$

$+W_a$, $+W_b$, $+B'_r$, $-W\}$ induces an $\overline{H_4}$ (A.11). Suppose, then, that $B'_r \not\longrightarrow F_r(B_l)$, thus $B_l \neq F_r(B_l)$. Then we check whether $W \longrightarrow F_r(B_l)$, i.e., whether $F_r(W) = F_r(B_l)$. If affirmative, then, as before, $W_b = U_l(B_l) \longrightarrow W$ is different to B_m and B'_r. Then, $\{v$, $+B_l$, $+F_r(B_l)$, $+B_m$, $+W_b$, $+B'_r$, $-W\}$ is a forbidden (A.12). When $W \not\longrightarrow F_r(B_l)$, we test if $W_a = F_l(B'_r) \longrightarrow W$ by looking if $F_l(W) = W_a$. If false, then $B'_r \neq W_a$ and $\{v$, $+B_l$, $+F_r(B_l)$, $+B_m$, $+W_a$, $+B'_r$, $-W\}$ is a forbidden (A.13). Otherwise, since W and B'_r are not indistinguishable, we obtain that $B'_r \not\longrightarrow F_r(W)$. Consequently, $F_r(W) \in (B_l, F_r(B_l))$ and $\{v$, $+B_l$, $+F_r(W)$, $+F_r(B_l)$, $+B_m$, $+B'_r$, $-W\}$ is a forbidden (A.14). Note that every computed forbidden is adequate by Observation 5.2.

By Claim 4, we may assume that every semiblock in (B_r, B_l) is indistinguishable to either B'_r or B_l. By (rec$_2$), (B_r, B_l) has at most two semiblocks, namely $R(B_r)$ and $L(B_l)$, that are indistinguishable to B_l and B_r. Then, two cases remain.

Case 1: $\#(B_r, B_l) = 2$. In this case, $\{B_l, L(B_l)\}$ and $\{B_r, R(B_r)\}$ are both pairs of indistinguishable semiblocks and, since B_l y B_r are not indistinguishable, then either $B_r \not\longrightarrow F_r(B_l)$ or $F_l(B_r) \not\longrightarrow B_l$. Suppose the former, as the other case is analogous. Then $B_l \neq F_r(B_l)$ and $\{v$, $-L(B_l)$, $+B_l$, $+F_r(B_l)$, $+B_m$, $+B_r$, $-R(B_r)\}$ is an adequate forbidden (A.15).

Case 2: $\#(B_r, B_l) = 1$. Applying Claims 1–4 to Φ^{-1}, we observe that $R(B_r)$ is indistinguishable to either B_r or $B'_l = \hat{U}_l^2(B_r)$. As $(B_r, B_l) \subseteq (B'_r, B'_l)$, we observe that either $B_r = B'_r$ is indistinguishable to $R(B_r)$ or $B_l = B'_l$ is indistinguishable to $R(B_r)$. Assume the former, as the other case is analogous. Now, if $\hat{U}_r(B_r) \not\longrightarrow F_r(B_m)$, then $\hat{U}_r(B_r) \neq B_m$ and $\{v$, $+B_l$, $+\hat{U}_r(B_r)$, $+B_m$, $+F_r(B_m)$, $+B_r$, $-R(B_r)\}$ is an adequate forbidden (possibly $B_m = F_r(B_m)$; A.16). Suppose, then, that $\hat{U}_r(B_r) \longrightarrow F_r(B_m) = \hat{L}(B_r)$, and let Ψ be the co-contig that is obtained from Φ by exchanging the order between B_r and $R(B_r)$. (For the rest of the proof, whenever we write f without a superscript we mean f^{Φ}.) Clearly, Ψ represents $H \setminus \{v\}$, $N(v) = [B_r, F_r(B_m)]$ in Ψ, while B_r and $R(B_r)$ are the only possible pair of indistinguishable semiblocks. That is, $\langle B_r, F_r(B_m)\rangle$ satisfies (rec$_1$)–(rec$_2$) for Ψ. Moreover, since $\hat{U}_r(B_r) \longrightarrow F_r(B_m)$, we obtain, by Claim 3 applied to Ψ, that $\langle B_r, F_r(B_m)\rangle$ satisfies (rec$_3$).

□

The main theorem of this section follows by combining Lemmas 5.6, 5.8, and 5.10 while filling the missing cases.

Theorem 5.11. *Let H be a graph with a non-universal vertex $v \in V(H)$, and Γ be block co-contig representing $H \setminus \{v\}$. Given Γ and $N(v)$, it costs $O(d(v) + \mathrm{ep}^+)$ time to determine if H is PCA. Furthermore, within the same amount of time, Γ can be transformed into a block co-contig representing H, unless a minimally forbidden of H is obtained.*

Proof. First suppose v is isolated in H. In this case there are three possibilities. First, if Γ is straight, then H is PCA and $\Gamma \cup \{\psi\}$ is a block co-contig representing H for any contig ψ whose only vertex is v. Second, if $\Gamma = \{\gamma\}$ is not straight and $|\mathcal{P}(\gamma)| \geq 4$, then $\mathcal{P}(\gamma) \cup \{v\}$ induces a cycle plus an isolated vertex. Thus, $\langle \mathcal{P}(\gamma), \emptyset\rangle$ is a minimally forbidden of H. Finally, if $\Gamma = \{\gamma\}$ is not straight and $\mathcal{P}(\gamma) = B_1, B_2, B_3$, then we can obtain a minimally forbidden by invoking Lemma 5.3 (b) with input B_1, B_2, B_3.

Now suppose $N(v) \neq \emptyset$. To compute a co-contig representing H, we first apply Lemma 5.6 with input $N(v)$ to verify that Γ has only good blocks. If negative, then we obtain a minimally forbidden. Otherwise, we apply Lemma 5.8 with input $N(v)$ to transform Γ into a block co-contig Γ' representing $H \setminus \{v\}$ in which $N(v)$ is consecutive. This time we obtain either a minimally forbidden or a pair of blocks $\langle B_a, B_b \rangle$ witnessing that $N(v)$ is consecutive in Γ'. In the latter case, we apply Lemma 5.10 with input $\langle B_a, B_b \rangle$ to transform Γ' into a co-contig Φ representing $H \setminus \{v\}$ that satisfies (rec$_1$)–(rec$_3$). By Lemma 3.2, we obtain either a minimally forbidden or a pair $\langle B_l, B_r \rangle$ that is v-receptive in $\{\phi, \phi_r\}$, where $\phi, \phi_r \in \Phi$ are the contigs that contain B_l and B_r, respectively. Finally, we check if $\langle B_l, B_r \rangle$ is receptive in Φ and we proceed as follows according to the answer.

Case 1: $\langle B_l, B_r \rangle$ is receptive in Φ. We first transform Φ into the $\{v\}$-reception Ψ of $\langle B_l, B_r \rangle$ in Φ. Then, we compact $\{v\}$ if it is indistinguishable to either $R^{\Psi}(\{v\})$ or $L^{\Psi}(\{v\})$. As a result, Ψ is a round block representation of H.

Case 2: $\langle B_l, B_r \rangle$ is not receptive in Φ. As discussed in Section 4.3, the only case in which $\langle B_l, B_r \rangle$ is receptive in $\{\phi, \phi_r\}$ and not receptive in Φ is when $\phi = \phi_r$, $(B_l, B_r]$ contains the left end semiblock W_l of ϕ, and $\Phi \setminus \{\phi\}$ has some contig ρ. Moreover, the v-reception $\{\psi\}$ of $\{\phi\}$ is not straight in this case. Thus, $H[V(\psi) \cup \{x\}]$ is not PCA for every $x \in V(\rho)$. Then, we can obtain a minimally forbidden of H by trying to insert a vertex $x \in V(\rho)$ into $\{\psi\}$. Since x has no neighbors in $V(\psi)$, this insertion trial costs $O(1)$ time, while we can find $x \in V(\rho)$ in $O(1)$ time by using the representation pointer of W_l.

As discussed in Section 4.3, $O(d(v) + \mathrm{ep}^+)$ time suffices to test if $\langle B_l, B_r \rangle$ is receptive in Φ and to compute the $\{v\}$-reception of $\langle B_l, B_r \rangle$ in Φ and $\{\phi\}$. By Lemmas 5.6, 5.8, and 5.10, we conclude that the whole algorithm costs $O(d(v) + \mathrm{ep}^+)$ time. □

5.2 H and G need not be co-connected

In this section we deal with the general case in which H and $G = H \setminus \{v\}$ need not be co-connected. In other words, $G = G[N] + G[V_1] + \ldots + G[V_k]$ where:

- V_u contains the universal vertices of G in $V(G) \setminus N(v)$, for some $1 \leq u \leq k$,
- For $i \neq u$, $G[V_i]$ is a co-component with $V_i \setminus N(v) \neq \emptyset$, and
- $N = V(G) \setminus (V_1 \cup \ldots \cup V_k)$ contains only vertices in $N(v)$.

We are taking a loose definition of $G[\bullet]$ and $+$ here, as it could happen that $V_u = \emptyset$, $N = \emptyset$, or $k = 1$; the missing details are obvious though. The algorithm in [30] builds a round block representation of H in two phases. The first phase finds a block co-contig ψ_v of $H[W_k \cup \{v\}]$, where $W_j = \bigcup_{i=1}^{j} V_i$ for every $0 \leq j \leq k$. The second phase joins ψ_v and a round block representation Γ_N of $H \setminus V$ into a round block representation Ψ of H. Our certifying algorithm mimics these two phases; the internal details are different, though.

The purpose of the first phase is to find a co-contig ψ_v of $H[W_k \cup \{v\}]$. To fulfill its goal, the algorithm in [30] computes all the round block representations of $H[W_k]$ to see if $N(v)$ is consecutive in one of them. For those in which $N(v)$ is consecutive, it checks if some of its v-associated representations is v-receptive. The algorithm is correct by the Reception Theorem and Observation 5.9, but it could require exponential time. A key observation in [30] is that H is not PCA when $k > 3$, thus only $O(1)$ round representations need to be examined, hence the algorithm is efficient. The problem with this "brute force" strategy is that it makes it difficult to find a negative certificate when H is not PCA. An alternative approach is to note that, as $H[W_k \cup \{v\}]$ is co-connected, at most two of the generated representations, Φ and Φ^{-1}, are v-receptive. The idea is to characterize how does Φ look like so that a minimally forbidden can be obtained when $H[W_k]$ has no v-receptive representations.

Instead of dealing with $H[W_k] = H[V_1] + \ldots + H[V_k]$ as a whole, we use an iterative approach. Before the algorithm is executed, we have a round block representation $\Gamma_0 = \Gamma$ of G and we build a new block co-contig Ψ_v of $H[\{v\}]$. After i iterations, we have transformed Γ into a round block representation Γ_i of $G \setminus W_i$ and ψ_v into a block co-contig of $H[W_i \cup \{v\}]$. To cope with iteration $i+1$, we use Steps 1–3 below. In brief terms, this procedure works a follows:

Step 1 splits from Γ_i a block co-contig γ_{i+1} having blocks co-adjacent to v. Let $V_{i+1} = V(\gamma_{i+1})$.

Step 2 updates γ_{i+1} into a block co-contig ψ_{i+1} of $H[V_{i+1} \cup \{v\}]$.

Step 3 joins ψ_{i+1} and ψ_v to obtain a block co-contig of $H[W_{i+1} \cup \{v\}]$.

Once the iterative process is completed, we have round block representations Γ_k of $G \setminus W_k$ and ψ_v of $H[W_k \cup \{v\}]$. We use Phase 2 below to combine these representations into a representation of H. Of course, any of these steps can fail, and a minimally forbidden is provided if so.

5.2.1 Step 1: split γ_{i+1} out of Γ_i

To split γ_{i+1} out of Γ_i, we traverse $\mathcal{B}(\Gamma_i)$ until the first block B co-adjacent to v is found. If no such block exists, then Phase 1 concludes and Phase 2 begins. Otherwise, we invoke Lemma 5.12 below to obtain the family E of co-end blocks of γ_{i+1}, where γ_{i+1} is the co-contig of Γ_i that contains B. If Lemma 5.12 outputs a minimally forbidden, then the algorithm halts; otherwise, we check if B is a universal block. If affirmative, then we separate B into $+B$ and $-B$, and we update γ_{i+1} to be the co-contig containing $-B$. The separation is done in $O(|+B|)$ time, as discussed in Section 4.3. Finally, we split γ_{i+1} out of Γ_i. Note that the case $E = \emptyset$ is trivial, as $\gamma_{i+1} = \Gamma_i$ and $\Gamma_{i+1} = \emptyset$, while the split when $E \neq \emptyset$ costs $O(1)$ time as discussed in Section 4.1. Therefore, Step 1 costs $O(d(v))$ time.

Lemma 5.12. *Let H be a graph with a vertex v, ϕ be a co-contig of a round representation Φ of $H \setminus \{v\}$, and $B \in \mathcal{B}(\phi)$ be co-adjacent to v. Given a Φ-pointer to B, it takes $O(d(v))$ time to determine if $G(\Phi)$ is co-bipartite when H is PCA. The algorithm outputs either a minimally forbidden of H or a set containing Φ-pointers to all the co-end semiblocks of ϕ.*

Proof. The algorithm outputs $\emptyset$ when Φ is not robust, and $\{\tilde{B}\}$ when B is universal. In the remaining case, the algorithm tries to locate the left co-end semiblocks of ϕ. For this, it computes the minimum $i \geq 0$ such that:

1. $\hat{U}_r^i(B)$ is a left co-end semiblock,
2. $\hat{U}_r^i(B) = \hat{U}_r^j(B)$ for some $j < i$, or
3. $i \geq 5$ and v is co-adjacent to $\hat{U}_r^{i-5}(B)$, $\hat{U}_r^{i-4}(B)$, $\hat{U}_r^{i-2}(B)$, and $\hat{U}_r^{i-1}(B)$.

Observe that $\hat{U}_r^i(B) \in \mathcal{B}(\phi)$ because B is not universal. Therefore: if 1. holds, then $\hat{U}_r^i(B)$ and $\hat{U}_r^{i+1}(B)$ are the left co-end semiblock of ϕ; if 2. holds, then $G(\Phi)$ is not co-bipartite because $\hat{U}_r^j(B), \ldots, \hat{U}_r^i(B)$ induces a co-cycle of odd length; and if 3. holds (and 2. does not), then H is not PCA because $\{v, -\hat{U}_r^{i-5}(B), -\hat{U}_r^{i-4}(B), -\hat{U}_r^{i-2}(B), -\hat{U}_r^{i-1}(B)\}$ induces a C_4^*. Clearly, i can be obtained in $O(d(v))$ time. Indeed, each semiblock is traversed $O(1)$ times by 1. and 2., while at most $6d(v)$ blocks co-adjacent to v are visited by 3. (See [30] for a better bound.) When 1. holds, the algorithm computes the right co-end semiblocks of ϕ by replacing $\hat{U}_r$ with $\hat{U}_l$ in 1–3. □

5.2.2 Step 2: update of γ_{i+1} into ψ_{i+1}

There are two possibilities for Step 2, according to whether γ_{i+1} has a unique (universal) block or not. In the former case, $\{v\}$ is a block of $H[V_{i+1} \cup \{v\}]$ co-adjacent to the clique V_{i+1}, thus computing the block co-contig ψ_{i+1} in $O(1)$ time is trivial. In the latter case, both $H[V_{i+1}]$ and $H[V_{i+1} \cup \{v_i\}]$ are co-connected. Thus, we invoke Theorem 5.11, with input γ_{i+1} and $V_{i+1} \cap N(v)$, to transform γ_{i+1} into a round block representation Ψ_{i+1} of $H[V_{i+1} \cup \{v\}]$. By Lemma 5.13, $O(d(v))$ time suffices to compute $V_{i+1} \cap N(v)$, thus Step 2 requires $O(d(v) + \text{ep}^+)$ time.

Lemma 5.13. *Let H be a graph with a vertex v, ϕ be a co-contig of a round representation Φ of $H \setminus \{v\}$, and B_l be a left co-end block of ϕ. Given $N(v)$ and a semiblock pointer to B_l, it takes $O(d(v))$ time to compute $V(\phi) \cap N(v)$ when H is PCA. When H is not PCA, the algorithm outputs either $V(\phi) \cap N(v)$ or a minimally forbidden of H.*

Proof. For each $B \in \mathcal{B}(\Phi)$ adjacent to v, we find a pointer $E(B)$ to a left co-end semiblock of the co-contig ϕ_B that contains B; initially, $E(B) = \bot$. To compute E, we traverse each $w \in N(v)$ to process the semiblock B that contains w. If B is universal, then we set $E(B) = B$ and pass to the next vertex. Otherwise, we look for the minimum $i \geq 0$ such that:

1. $E(\hat{U}_r^i(B)) \neq \bot$,
2. $\hat{U}_r^i(B)$ is a left co-end semiblock, or
3. $i \geq 4$ and v is co-adjacent to $\hat{U}_r^{i-4}(B)$, $\hat{U}_r^{i-3}(B)$, $\hat{U}_r^{i-1}(B)$, and $\hat{U}_r^i(B)$.

Since B is not universal, it follows that B and $\hat{U}_r^j(B)$ belong to the same co-component for every $0 \leq j \leq i$. Hence, $E(\hat{U}_r^i(B))$ is a left co-end semiblock of ϕ_B if 1., while $\hat{U}_r^i(B)$ is a left co-end semiblock of ϕ_B if 2. Therefore: if 1., then we set $E(\hat{U}_r^j(B)) = E(\hat{U}_r^i(B))$ for every $0 \leq j \leq i$; if 2., then we set $E(\hat{U}_r^j(B)) = \hat{U}_r^i(B)$ for every $0 \leq j \leq i$; and if 3., then we output that H is not PCA because $\{v, -\hat{U}_r^{i-4}(B), -\hat{U}_r^{i-3}(B), -\hat{U}_r^{i-1}(B), -\hat{U}_r^i(B)\}$ induces a C_4^*. The computation of E ends after all the vertices in $N(v)$ have been considered. After E is computed, the algorithm outputs $V(\phi) \cap N(v) = \{w \in N(v) \mid E(B(w)) \in \{B_l, U_r(B_l)\}\}$, where $B(w)$ is the semiblock that contains w. Clearly, by 1. and 2., the algorithm traverses each semiblock B adjacent to v only $O(|B \cap N(v)|)$ times, while, by 3., it traverses at most $5d(v)$ blocks co-adjacent to v. □

5.2.3 Step 3: join of ψ_{i+1} and ψ_v

Step 3 has to join ψ_v and ψ_{i+1} into a block co-contig representing $H[W_{i+1} \cup \{v\}]$. This is trivial when $i = 1$ as we replace ψ_v with ψ_1. When $i > 1$, at most one between $H[W_i]$ and $H[V_{i+1}]$ is a clique. Thus, we can combine ψ_v and ψ_{i+1} in $O(1)$ time with the following lemma.

Lemma 5.14. *Let H be a co-connected graph with a vertex v such that $H \setminus \{v\}$ is a PCA graph isomorphic to $H[V_1] + H[V_2]$ for some $\emptyset \subset V_1 \subset V(H)$ and $V_2 = V(H) \setminus (V_1 \cup \{v\})$, and let B_i be the block that contains v in a block co-contig ψ_i representing $H[V_i \cup \{v\}]$, for $i \in \{1, 2\}$. Suppose V_1 is not a block of ψ_1. Then, H is a PCA graph if and only if either:*

(i) *B_1 and B_2 are co-end blocks of ψ_1 and ψ_2,*

(ii) *V_2 is a block of ψ_2, ψ_1 is robust, and $F_r(\hat{R}(B_1)) = U_l(\hat{L}(B_1)) \neq \bot$, or*

(iii) *V_2 is a block of ψ_2 and V_1 has exactly three non-adjacent blocks: $\{v, W_2, W_3\}$.*

Consequently, $O(1)$ time suffices to determine if H is PCA, when ψ_i-pointers to B_i are given. The algorithm either transforms ψ_1 and ψ_2 into a block co-contig representing H or outputs a minimally forbidden of H.

Proof. First we prove that H is a PCA graph when some of (i)–(iii) holds.

(i) holds. The proof is implicit in [30]. By reversing ψ_1 and ψ_2 if required, suppose B_1 is a right co-end block and B_2 is a left co-end block. As discussed in Section 4.3, we can join ψ_1 and ψ_2 into a block co-contig ρ representing $G(\rho) = G(\psi_1)+G(\psi_2) = H[V_1 \cup \{v\}]+H[V_2 \cup \{v\}]$ in which B_1 and B_2 are consecutive. Clearly, B_1 witnesses that $\langle F_l(B_1), F_r(B_2)\rangle$ is receptive in ρ, thus the $\{w\}$-reception of $\langle F_l(B_1), F_r(B_2)\rangle$ is a block co-contig representing a graph H' with three vertices $v_1 \in B_1$, $v_2 \in B_2$ and w such that: $N(w) = N_H(v)$ and $H' \setminus \{v_1, v_2, w\} = H \setminus \{v\}$. Consequently, $H = H' \setminus \{v_1, v_2\}$ is PCA.

(ii) holds. Let ρ be the co-contig that is obtained from ψ_1 by a separation of B_1 into $\langle B_1 \setminus \{v\}, \{v\}\rangle$, and note that $B_m = F^\rho_r(\hat{R}^\rho(\{v\}))$ witnesses that $\langle \hat{R}^\rho(\{v\}), \hat{L}^\rho(\{v\})\rangle$ is receptive in ρ. Consequently, the V_2-reception of $\langle \hat{R}^\rho(\{v\}), \hat{L}^\rho(\{v\})\rangle$ is a block co-contig that represents H because v is the unique vertex not adjacent to V_2.

(iii) holds. Trivial.

It is not hard to obtain the block co-contig ρ in $O(1)$ time using the operations described in Section 4.3 with some low-level manipulation of the contigs (i.e., avoiding `reception`); see [30].

Now suppose none of (i)–(iii) holds. To prove that H is not PCA we show an $O(1)$ time algorithm that computes a minimally forbidden of H. If B_2 is not a co-end block, then V_2 is not a block, thus we may replace V_1 and V_2 without affecting the hypothesis of the lemma. Hence, as (i) is false, we suppose B_1 is not a co-end block. Through the proof we work only with ψ_1 and two blocks of ψ_2, called X and Y, which are not adjacent to B_2 and X, respectively. Also, $Y \neq \{v\}$ unless $V_2 = X$. Note that X and Y are obtainable in $O(1)$ time.

The first step of the algorithm is to verify if ψ_1 is robust. By Theorem 2.1, $H[V_1]$ is co-bipartite because $H \setminus \{v\}$ is not co-connected. Then, $\psi_1 \setminus \{B_1\}$ is robust, thus either ψ_1 is robust or $B_1 = \{v\}$ is isolated in ψ_1 and $\psi_1 \setminus \{B_1\}$ has exactly two non-adjacent blocks W_2, W_3. Therefore, we can decide if ψ_1 is robust in $O(1)$ time, obtaining pointers to W_2 and W_3 if negative. Moreover, $Y \neq \{v\}$ because (iii) is false. Consequently, $\{B_1, X, W_2, W_3, Y \setminus \{v\}\}$ contains either a $K_{1,3}$ or a C_4^*. Such a minimally forbidden can be obtained in $O(1)$ time. From now on we assume ψ_1 is robust, hence $\hat{L}$, $\hat{R}$, $\hat{U}_l$, and $\hat{U}_r$ are well defined for ψ_1. Moreover, as B_1 is not a co-end block, we obtain that $\hat{U}_r(B_1) \not\twoheadrightarrow \hat{L}(B_1)$ and $\hat{R}(B_1) \not\twoheadrightarrow \hat{U}_l(B_1)$.

The second step is to check if $\hat{U}_l(B_1) \xrightarrow{=} \hat{L}(B_1)$ and if $\hat{R}(B_1) \xrightarrow{=} \hat{U}_r(B_1)$. If $\hat{U}_l(B_1) \neq \hat{L}(B_1)$ and $\hat{U}_l(B_1) \not\twoheadrightarrow \hat{L}(B_1)$, then $\hat{L}(B_1) \not\twoheadrightarrow F_r(B_1)$ because B_1 and $\hat{L}(B_1)$ are not indistinguishable. So, $F_r(B_1) \longrightarrow \hat{U}_l(B_1)$ because, otherwise, $\hat{U}_l(B_1)$, $\hat{L}(B_1)$, $F_r(B_1)$ are pairwise non-adjacent blocks, contradicting the fact that $H[V_1]$ is co-bipartite. Similarly, $\hat{L}(B_1) \longrightarrow \hat{R}(B_1)$ because $\hat{U}_l(B_1)$, $\hat{L}(B_1)$, $\hat{R}(B_1)$ cannot be pairwise non-adjacent. Hence, B_1, $\hat{R}(B_1)$, $F_r(B_1)$, $U_r(B_1)$ are pairwise different and appear in this order in a traversal of $[B_1, \hat{U}_r(B_1)]$. The minimally forbidden we generate depends on whether $\hat{R}(B_1) \longrightarrow \hat{U}_r(B_1)$ or not. In the affirmative case, $\{B_1$, $\hat{L}(B_1)$, $\hat{R}(B_1)$, $F_r(B_1)$, $\hat{U}_r(B_1)$, $\hat{U}_l(B_1)$, $X\}$ induces an $\overline{H_4}$ (A.17). In the negative case we observe that, as before, $F_l(B_1) \not\twoheadrightarrow \hat{R}(B_1)$ and $F_r(B_1) \longrightarrow \hat{U}_l(B_1)$. This implies $F_l(B_1) \longrightarrow F_r(B_1)$ because $\{F_l(B_1)$, $\hat{L}(B_1)$, $\hat{R}(B_1)$, $F_r(B_1)$, $\hat{U}_r(B_1)\}$ does not induce a C_5, thus $\{B_1$, $\hat{L}(B_1)$, $\hat{R}(B_1)$, $F_r(B_1)$, $F_l(B_1)$, $\hat{U}_l(B_1)$, $X\}$ induces an $\overline{H_5}$ (A.18). From now on, we assume $\hat{U}_l(B_1) \xrightarrow{=} \hat{L}(B_1)$ and, similarly, $\hat{R}(B_1) \xrightarrow{=} \hat{U}_r(B_1)$. Hence, $\hat{U}_l(B_1) \neq \hat{U}_r(B_1)$.

Note that $F_r(\hat{R}(B_1)), F_l(\hat{L}(B_1))$ are either equal or appear in this order in a traversal of $[B_1, \hat{L}(B_1)]$; otherwise, any block inside $(F_l(\hat{L}(B_1)), F_r(\hat{R}(B_1)))$ would be indistinguishable to

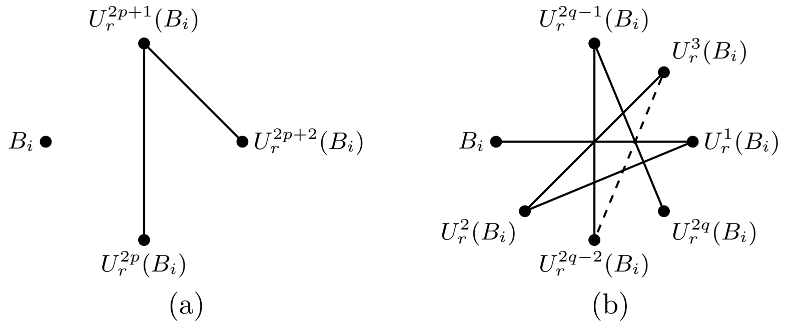


Figure 8: Adjacencies of $\overline{H[V_i]}$ in Theorem 5.15. The blocks are drawn as they appear in the circular ordering $\mathcal{B}(\Phi_i)$. Note that $U_r^{2q}(B_i) = U_r^{2q-2}(B_i)$ when $H[V_i]$ is co-bipartite.

$F_r(\hat{R}(B_1))$. For the third step, the algorithm tests if $(F_r(\hat{R}(B_1)), F_l(\hat{L}(B_1)))$ has some block W. If affirmative, then $\hat{L}(B_1) \longrightarrow \hat{R}(B_1)$ since, otherwise, W, $\hat{L}(B_1)$, $\hat{R}(B_1)$ are pairwise non-adjacent. Consequently, $\hat{L}(B_1)$, $\hat{R}(B_1)$, $\hat{U}_r(B_1)$, and $\hat{U}_l(B_1)$ are all different. This implies that $\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)$ because no subset of $\{W, \hat{U}_l(B_1), \hat{L}(B_1), \hat{R}(B_1), \hat{U}_r(B_1)\}$ induces an $\overline{C_3}$ or C_5. Consequently $\{B_1, X, W, \hat{U}_l(B_1), \hat{U}_r(B_1), \hat{L}(B_1), \hat{R}(B_1)\}$ induces an $\overline{H_2}$ (A.19).

Finally, note that $Y \neq \{v\}$ when either $F_r(\hat{R}(B_1)) = F_l(\hat{L}(B_1))$ or $F_r(\hat{R}(B_1)) = \hat{U}_l(\hat{L}(B_1))$. Indeed, in the former case $Y \neq \{v\}$ because $F_r(\hat{R}(B_1))$ and X are not twins, while in the latter case $Y \neq \{v\}$ because (ii) is false. Consequently, $\mathcal{B} = \{B_1, Y \setminus \{v\}, X, \hat{U}_l(B_1), \hat{L}(B_1), \hat{R}(B_1), \hat{U}_r(B_1)\}$ is a forbidden (A.20) whose edges can be obtained in $O(1)$ time. We remark that not all the blocks in $\mathcal{B}$ are pairwise different. □

5.2.4 Phase 2: join of Ψ_v and Γ

After the first phase is completed, we have a round block representation Γ of $G \setminus W_k$ and a block co-contig ψ_v representing $H[W_k \cup \{v\}]$ for $W_k = \bigcup_{i=1}^{k} V_i$. The goal of the second phase is to find a round block representation of H. This is trivial when $W_k = V(G)$, as ψ_v is the desired representation. For the other case, we invoke Theorem 5.15 using v and $w \in V(\Gamma)$ as input.

Theorem 5.15. *Let H be a graph such that $H = H[V_1] + H[V_2]$ for $\emptyset \subset V_1, V_2 \subset V(H)$, and Φ_i be a round block representation of $H[V_i]$, for $i \in \{1, 2\}$. Then, H is PCA if and only if $H[V_1]$ and $H[V_2]$ are PCA and co-bipartite. Furthermore, if semiblock Φ_i-pointers to $B_i \in \mathcal{B}(\Phi_i)$ are given, then $O(|N(B_i)|)$ time suffices to determine if H is PCA. The algorithm either transforms Φ_1 and Φ_2 into a round block representation of H or outputs a minimally forbidden of H.*

Proof. The fact that H is PCA if and only if $H[V_1]$ and $H[V_2]$ are co-bipartite PCA graphs follows from Theorem 2.1.

The algorithm to detect if H is PCA is as follows. Let B_i be any block of Φ_i ($\{i, j\} = \{1, 2\}$), and q be the minimum such that either $U_r^{2q}(B_i) \twoheadrightarrow B_i$ or $U_r^{2q}(B_i) = U_r^{2q-2}(B_i)$. Note that, since $U_r^{2p}(B_i) \overset{=}{\longrightarrow} B_i$, the blocks $B_i, U_r^{2p+1}(B_i), U_r^{2p+2}(B_i), U_r^{2p}(B_i)$ appear in this order in $\mathcal{B}(\Phi_i)$ for every $0 \leq p < q$ (see Figure 8 (a)). Consequently, the value q is well defined, and the blocks of Φ_i appear as in Figure 8 (b). Therefore, if p is the maximum such that $U_r^{2q}(B_i) \twoheadrightarrow U_r^{2p}(B_i)$, then either $p = q - 1$ or $\mathcal{B} = \{U_r^{2p}(B_i), \ldots, U_r^{2q}(B_i)\}$ induces an odd co-cycle. In the former case, $U_r^{2q}(B_i) = U_r^{2q-2}(B_i)$ is a co-end block, while, in the latter case, $\mathcal{B} \cup \{B_j\}$ is a minimally forbidden of H for every $B_j \in \mathcal{B}(\Phi_j)$. Replacing i by j, we can find a minimally forbidden when Φ_j has no co-end blocks. When both Φ_1 and Φ_2 have co-end blocks, we can `join` Φ_1 and Φ_2 into a round block representation of H as in Section 4.3.

To compute the sequence $B_i, \ldots, U_r^{2q}(B_i)$ we proceed as follows. First, we mark all the blocks in $[B_i, F_r(B_i)]$. Then, $U_r^{2i}(B_i) \longrightarrow B_i$ if and only if $F_r(U_r^{2i}(B_i))$ is marked; thus, q is the minimum value such that $F_r(U_r^{2q}(B))$ is not marked. Then, to obtain the value p, first note that $p = 0$ if $q = 1$. When $q > 1$, we traverse $[F_l(B_i), B_i]$ while looking for $F_r(U_r^{2q}(B_i))$; then $U_r^{2p+2}(B_i)$ is the last block of the traversed sequence. Since B_i and B_j are adjacent to every block in $[F_l(B_i), F_r(B_i)]$, the cost of this algorithm is $O(\min\{|N(B_i)|, |N(B_j)|\})$. □

6 The certifying recognition algorithms

By Theorem 2.1, at most three iterations of Phase 1 in Section 5 can be completed without finding a minimally forbidden. Hence, since each iteration of Phase 1 costs $O(d(v) + \mathrm{ep}^+)$ time, and Phase 2 costs $O(d(v))$ time, we obtain the main result of the previous section: there is an $O(d(v) + \mathrm{ep}^+)$ algorithm that transforms a round block representation Γ of $H \setminus \{v\}$ into a round block representation Ψ of H, unless a minimally forbidden is obtained. Note that the algorithm ignores the straightness invariant of Γ, and it does not ensure the straightness invariant for Ψ. The straightness invariant, instead, is required for the recognition of PIG graphs. Fortunately, we can restore the straightness invariant in $O(1)$ time with Corollary 6.4 below. Before describing this corollary, we define what a locally straight representation is.

Recall that a semiblock B of a round representation Φ is long when $F_r(B) \longrightarrow F_l(B)$. When no block of Φ is long, Φ is said to be a *locally straight representation.* A graph G is a *proper Helly circular-arc (PHCA)* graph if it is isomorphic to $G(\Phi)$ for some locally straight representation Φ. As it is shown in [24], G is a PHCA graph if and only if it admits a PCA model in which no two nor three arcs cover the circle. The following results imply Corollary 6.4 below.

Theorem 6.1 ([24])**.** *A PCA graph is a PHCA graph if and only if it contains no W_4 or S_3 as induced subgraphs, where W_4 is the graph obtained after inserting a universal vertex in C_4.*

Theorem 6.2 ([24])**.** *If B is the universal block of a contig ϕ, then either 1. ϕ is linear, 2. $F_r(L(B)) = B$ or 3. $G(\phi)$ is not PHCA. If 2., then $F_r(B)$ witnesses that $\langle R(B), L(B)\rangle$ is receptive in $\phi \setminus \{B\}$, and its B-reception is a linear contig representing $G(\phi)$.*

Lemma 6.3 ([24])**.** *If a round representation Φ has three non-universal blocks B_1, B_2, B_3 such that $B_1 \longrightarrow B_2$, $B_2 \longrightarrow B_3$, and $B_3 \longrightarrow B_1$, then $G(\Phi)$ is not PHCA.*

Corollary 6.4. *Given a round block representation Ψ, it takes $O(1)$ time to transform Ψ into a round block representation Ψ' of $G(\Psi)$ that satisfies the straightness invariant. Moreover, Ψ' is locally straight when $G(\Psi)$ is PHCA.*

Proof. By Theorems 2.2, 6.1, 6.2, and Lemma 6.3, the algorithm has nothing to do in the following situations because either 1. Ψ is straight, 2. Ψ is locally straight and $G(\Psi)$ has an induced cycle, or 3. $G(\Psi)$ is not PHCA:

- $|\Psi| > 1$,
- $\Psi = \{\psi\}$ and $|\mathcal{P}(\psi)| > 3$,
- $\Psi = \{\psi\}$, $|\mathcal{P}(\psi)| = 3$ and no block of $\mathcal{P}(\psi)$ is universal, or
- $\Psi = \{\psi\}$, $|\mathcal{P}(\psi)| = 3$, $B \in \mathcal{P}(\psi)$ is universal, and $F_r(L(B)) \neq B$.

Finally, if $\Psi = \{\psi\}$, $|\mathcal{P}(\psi)| = 3$, $B \in \mathcal{P}(\psi)$ is universal, and $F_r(L(B)) = B$, the algorithm moves B to the position that follows $F_r(B)$ in a traversal of $\mathcal{B}(\Psi)$. The block representation Ψ' so obtained is straight by Theorem 6.2. Clearly, $O(1)$ time is enough to test the above conditions and to apply the required move using `split` and `join` (see Section 4.3). □

The main theorems of this article then follow.

Theorem 6.5. *Let H be a graph with a vertex v, and Γ be a round block representation of $H \setminus \{v\}$. Given Γ and $N(v)$, it takes $O(d(v) + \mathrm{ep}^{+})$ time to determine if H is a PCA graph. The algorithm transforms Γ into a round block representation of H that satisfies the straightness invariant, unless a minimally forbidden of H is obtained.*

Theorem 6.6. *When a vertex v of a round block representation Ψ is given, $O(d(v) + \mathrm{ep}^{-})$ time is enough to transform Ψ into a round block representation of $G(\Psi) \setminus \{v\}$ that satisfies the straightness invariant.*

Proof. Let B be the block that contains v. If $|B| > 1$, then we remove v out of B; otherwise we call `remove(B)` to transform Ψ into a round block representation Φ of $H \setminus \{v\}$. Afterwards, we apply Corollary 6.4 on Φ to restore the straightness invariant. □

Theorem 6.7. *Given a round block representation Γ of a graph H that satisfies the straightness invariant, it takes $O(1)$ time to determine if H is PHCA. If H is not PHCA, then the algorithm outputs $\Gamma|\mathcal{B}$ for a family of blocks $\mathcal{B}$ such that $H[\mathcal{B}]$ is isomorphic to either W_4 or S_3.*

Proof. The algorithm answers yes when $|\Gamma| > 1$ or $\Gamma = \{\gamma\}$ and $|\mathcal{P}(\gamma)| > 3$. Conversely, if $\Gamma = \{\gamma\}$ and $\mathcal{P}(\gamma) = B_1, B_2, B_3$, then, by the straightness invariant, H is not PHCA. Moreover, $H[\mathcal{B}]$ is not PHCA for $\mathcal{B} = \{B_1, B_2, B_3, U_r(B_1), U_r(B_2), U_r(B_3)\}$ [24]. As discussed in Lemma 5.3, we can compute all the adjacencies of $H[\mathcal{B}]$ in $O(1)$ time. □

Theorem 6.8. *Given a round block representation Γ of a graph H that satisfies the straightness invariant, it takes $O(1)$ time to determine if H is PIG. If H is not PIG, then the algorithm outputs $\Gamma|\mathcal{B}$ for a family of blocks $\mathcal{B}$ such that $H[\mathcal{B}]$ is either a S_3 or a C_k ($k \geq 4$).*

Proof. By the straightness invariant, all we need to do to test if H is PIG is to call `straight` (Section 4.3). If H is not straight, then we test if H is PHCA. If negative, then we extract an induced S_3 or C_4 from the output $\Gamma|\mathcal{B}$. Otherwise, $\Gamma = \{\gamma\}$ and $\mathcal{P}(\gamma)$ induces a C_k ($k \geq 4$). □

6.1 The authentication problems

As discussed in Section 1, we can conceive three types of checkers, namely static, dynamic, and monitors. The static checker, which has the simplest implementation, authenticates the witnesses against the static graph G. The dynamic checker, instead, test the witness obtained after one operation is applied on a round block representation Φ. Although it is more efficient than applying the static checker for each update, the dynamic checker requires some extra effort as different tests are performed for the different updates. Finally, the monitor is a new layer between the end user and the dynamic algorithm that checks the correct behavior of Φ and the witnesses it generates. To do its work in the most efficient way, the monitor requires privileged access to some operations that are restricted to the final user [2]. Thus, the implementation of the monitor is not as simple as for the checkers, as it requires some knowledge about the internal representation of Φ. In this section we briefly discuss the static and dynamic checkers, and we skim through a possible design of a monitor.

6.1.1 The static checker

The static checker authenticates that a witness W is correct for a graph G. Of course, the correctness depends on the recognition problem we are dealing with and on whether W is positive or negative. Since we consider three problems, i.e., the recognition of PIG, PHCA, and PCA graphs, the static checker has to solve six problems.

When G is claimed to be PCA, the witness is a round block representation Φ of G. To authenticate Φ, the checker tests if:

- each $\phi \in \Phi$ is a block contig,
- $G(\Phi)$ is isomorphic to G,
- every block of G corresponds to a block of Φ, and
- if required, Φ is (locally) straight.

On the other hand, if G is declared as not being a member of the class, then the negative witness is a minimally forbidden $\langle \Phi, \mathcal{N} \rangle$, where $\mathcal{N} = \emptyset$ when the problem is the recognition of PIG or PHCA graphs. To authenticate that $\langle \Phi, \mathcal{N} \rangle$ is correct, the checker builds the graph F represented by $\langle \Phi, \mathcal{N} \rangle$, and then it tests that F is isomorphic to an induced subgraph of G.

It is not hard to see that these problems can be solved in $O(n + m)$ time. Moreover, the implementation of the checker is simple as desired.

6.1.2 The dynamic checker

The static checker is optimal for the authentication of static graphs. However, its time complexity is excessive when compared to the time required by an update of the round block representation Γ. Thus, the static checker is not well suited for the dynamic algorithm. The dynamic checker, instead, tries to authenticate the witness against Γ. Of course, the authentication depends on the applied update and the kind of witness obtained.

Suppose we want to authenticate a successful insertion. In this case, the input is $N(v)$ and a round block representation Γ. The output is the vertex v, and the witness is the round block representation Ψ of G that satisfies the straightness invariant. Let B be the block of Ψ that contains v. To authenticate Ψ, we check that:

(i) Ψ is a round block representation that satisfies the straightness invariant,

(ii) $N(v) = \bigcup [F_l(B), F_r(B)]$,

(iii) $G(\Psi) \setminus \{v\}$ is isomorphic to $G(\Gamma)$, and

(iv) the vertices of $G(\Gamma)$ appear in the same blocks in $\Psi \setminus \{\{v\}\}$ and Γ.

There is a large asymmetry between the `insert` operation and its authentication. Whereas the former deals mostly with $N(v)$, the latter tests all the blocks of Ψ. There are three reasons why the checker must look at the complete structure. First, because the dynamic algorithm could modify $O(n)$ far neighbors even when $d(v) = O(1)$ (e.g., when the universal block $B \in \Gamma$ is separated into $-B$ and $+B$). Second, and most important, because we cannot assure that a buggy implementation of `insert` updates only the blocks it is supposed to. Third, because the specification of `insert` requires $G(\Gamma)$ to be isomorphic to $G(\Psi) \setminus \{v\}$, and no more guaranties are provided.

It is not hard to see that (i) and (ii) can be implemented in $O(n)$ time. For (iii) and (iv), the checker works as follows. Let $\Phi = \Psi \setminus \{\{v\}\}$. First, the checker looks for all the co-end blocks in both Φ and Γ. Note that Φ needs not be a block representation. However, it is not hard to consider the twin semiblocks of Φ as being part of the same block; we omit the details. Also, observe that Φ is not actually computed; instead v is ignored in Ψ. In the second step, the algorithm checks that the co-end blocks of Φ and Γ coincide. If not, the checker reports that the implementation is buggy. When all the co-end blocks coincide, the checker traverses each co-contig ϕ of Φ to test that the remaining blocks appear in the same order as in Γ (or its reversal). If negative, then the checker outputs that the implementation is buggy; otherwise, both (iii) and (iv) hold. The correctness of this algorithm follows from the fact that co-connected PCA graphs admit exactly two round block representations, one the reverse of the other [16]. Note that the dynamic checker for the insertion runs in $O(n)$ time. Its implementation, however, is not as simple as the one for the static checker.

The authentication required for `remove` is similar and can be implemented in $O(n)$ time as well. Analogously, the authentication that Φ is either straight or locally straight, required for `forbiddenPIG` and `forbiddenPHCA`, takes $O(n)$ time. Finally, to verify a negative witness $\langle \Phi, \mathcal{N} \rangle$, the checker tests that Φ is indeed a representation induced from Γ, and that $\mathcal{N}$ contains the neighbors of v in Φ. Both of these tests can be easily implemented in $O(n)$ time.

6.1.3 The monitor

Although the dynamic checker is much faster than the static one, it is still too expensive when compared to the update operations. Unfortunately, the dynamic checker is optimal when no details about the implementation can be exploited. When we have access to the implementation of the data structure, we can *monitor* each operation to ensure its correctness [2, 25]. Recall that the dynamic algorithm deals with five data structures, namely contigs, semiblock paths, round representations, connectivity structures, and witnesses. The idea is to implement these data types in a way that we can trust all of them.

To make the above statement more precise, consider the `separate` operation of contigs. Recall that `separate(`$\tilde{B}$`, `W`)` transforms the contig γ referenced by $\tilde{B}$ into the contig ϕ that represents $G(\phi)$ by splitting B into two indistinguishable semiblocks $B \setminus W$ and W in such a way that $R^{\phi}(W) = R^{\gamma}(B)$, $L^{\phi}(W) = B \setminus W$, and $L^{\phi}(B \setminus W) = L^{\gamma}(B)$. To verify that `separate` is correct, a checker must guarantee, among other things, that ϕ represents $G(\gamma)$. There are at least two inconveniences that the checker must confront. First, a buggy implementation could fail to update F_r for some neighbor of B. Second, there could be $O(n)$ semiblocks that have B as its right far neighbor in Φ, and all of them should reference W in Γ. Thus, if the data structure is unknown, then the checker must traverse $O(n)$ semiblocks to authenticate γ. However, the implementation spends $O(1)$ time to simultaneously update all the right far neighbors. In fact, the algorithm consists of swapping two *self* pointers [13, 30]. If we were given access to the self pointers, then we could test that the swap is correct. A second and more pragmatic approach is to consider that such a swap is correct by definition. The reason is that proving the correctness of an implementation of swap is as simple, if not simpler, than authenticating the output of swap. Moreover, if we cannot trust the implementation of swap, then we cannot trust the implementation of the monitor either.

In a similar way as described for the update of far pointers, we may assume that a contig ϕ provides other basic operations, which are accessible only to the monitor, that are correct by definition. However, some operations are harder to implement and should be monitored. We differentiate three types of errors that impact on the design of ϕ and its monitor.

(Improper) access errors arise when a portion of the data structure that should not be accessed is modified. For instance, only the semiblocks in $[B_l, B_r]$ need to be updated in `reception(`B_l`, `B_r`)`. We consider those modifications to semiblocks outside $[B_l, B_r]$ as access errors. There are at least two basic methods for dealing with access errors. The simplest one is to ignore the error; this strategy is appropriate if we can assure that the error will be caught when the modified portion of the structure is accessed. The alternative method is to use some kind of supervised memory that tracks all the updates of the data structure. Then, the monitor can refuse those operations that access a restricted portion of the memory. The first approach is used in [25] for ordered dictionaries. When the monitor asks the dictionary to `insert` a pair (k, i), the dictionary could (erroneously) erase an item (k', i'). Such misbehavior is not detected by the monitor until it tries to access (k', i'). Thus, the monitor is not certifying the whole data structure for `insert`, but only that the insertion takes place where it should. We remark that missing such an error is not critical for dictionaries, because the elements that it holds are independent of each other. For contigs, perhaps it is better to take actions immediately using the supervised memory solution.

Memory errors occur when an uncontrolled memory location is accessed. To deal with uncontrolled memory locations, we can follow the same technique as in [25]. That is, each semiblock $B \in \Phi$ keeps the position of a semiblock pointer $\tilde{B}$ in an array T of "trusted" memory. This array is controlled by the monitor to ensure that each access to B is correct. To authenticate the access to B, the monitor access its position of T and uses $\tilde{B}$ to control that B was under the control of the data structure.

Logical errors happen when an operations does not behave as it is supposed to, but accessing only the portions of the data structure to which they have access. Suppose, for instance, that the monitor is asked to perform `reception(`B_l`,` B_r`)` on ϕ. The monitor forwards this operation to the data structure and it obtains the semiblock B that contains v on ψ. When B is not an end semiblock, the monitor outputs that the implementation is buggy if some of the following check fails.

- $F_l(B) = B_l$ and $F_r(B) = B_r$,
- $F_r^{\psi}(W) = B$ for every $W \in [B_l, B)$ such that $F_r^{\phi}(W) = L^{\psi}(B)$,
- $F_r^{\psi}(W) = F_r^{\phi}(W)$ for every $W \in [B_l, B)$ such that $F_r^{\phi}(W) \neq L^{\psi}(B)$,
- $F_l^{\psi}(W) = \{v\}$ for every $W \in (B, B_r]$ such that $F_l^{\phi}(W) = R^{\psi}(\{v\})$, and
- $F_l^{\psi}(W) = F_l^{\phi}(W)$ for every $W \in (B, B_r]$ such that $F_l^{\phi}(W) \neq R^{\psi}(\{v\})$.

The case in which B is an end semiblock is handled similarly.

Using the above techniques, we can authenticate all the operations on contigs. Then, the remaining data types should be monitored as well. Suppose we need to check that `reception(`B_l`,` B_r`)` works as specified for a round representation Φ. A priori, the only operation of contigs that should be invoked is the trusted `reception` with inputs B_l and B_r. Thus, any other update on the contigs should be regarded as an access error. Following the supervised memory solution, we may ask the monitor of contigs to track the updates that it performs. Then, the monitor of Φ can observe that the only update on its contigs was the `reception` of B_l and B_r. Since this operation is under supervision, we many assume it is correct, thus we only need to check the logical errors. In this case, that the obtained contig is not circular when $|\Psi| > 1$ for the obtained round representation.

7 Conclusions

We designed a new dynamic algorithm for the recognition of PCA, PHCA, and PIG graphs that allows vertex updates. The algorithm keeps a round block representation Φ of the input graph G that can be regarded as being a positive witness. When the insertion of v into $G = H \setminus \{v\}$ fails, the algorithms exhibits a minimally forbidden induced subgraph F of H. To work as fast as possible, the algorithm keeps a partial view of $F \setminus \{v\}$ that contains all but $O(d(v))$ vertices of F. The problem of finding a negative certificate when edges updates are allowed is left open.

The certifying algorithm is optimal when applied for the recognition of static graphs, as it runs in $O(d(v))$ time per inserted vertex. The algorithm is almost optimal when both insertions and removal are allowed, as it requires $O(d(v) + \log n)$ time per operation and the lower bound in the cell probe model of computation with word-size b is $\Omega(d(v) + \log n/(\log\log n + \log b))$ [13].

Regarding the authentication problem, we considered three possibilities, each one giving rise to a different kind of checker. Static checkers test the result of the algorithm for a static graph G. Its input is G together with either a round representation Φ or a graph F, and the goal is to verify that Φ is a round block representation of G or that F is a minimally forbidden induced subgraph of G. Dynamic checkers, instead, test that an operation on a round block representation Φ is successful. Its input, then, is Φ plus the input of the operation and either a round representation Ψ or a minimally forbidden F. The goal in this case is to verify that Ψ is a round block representation of the graph H that should be obtained from $G(\Phi)$ when the operation is applied or to test that F represents a minimally forbidden induced subgraph of H. By definition, the problems associated to the static and dynamic checkers are static and require $\Omega(n + m)$ and $\Omega(n)$ time in the worst case as either G or Φ have to be traversed, respectively. Monitors, instead, are dynamic algorithms (i.e., data structures) that sit between the user and the round block representation Φ of the dynamic graph G. When a monitor has access to some privileged (query) operations on Φ, the time required for the authentication can be reduced. In this article we skim through the process of designing a monitor for the algorithm which, we believe, could be used to authenticate each operation in $O(t)$ time, where t is the time required by the operation itself. There is no proof of this fact, as the monitor is incomplete; yet, we discussed some issues that can arise when such a monitor is developed.

References

[1] J. Bang-Jensen and G. Gutin. *Digraphs.* Springer Monographs in Mathematics. Springer-Verlag London, Ltd., London, second edition, 2009. doi: 10.1007/978-1-84800-998-1.

[2] J. D. Bright and G. F. Sullivan. On-line error monitoring for several data structures. In *Digest of Papers: FTCS-25, The Twenty-Fifth International Symposium on Fault-Tolerant Computing, Pasadena, California, USA, June 27-30, 1995*, pp. 392–401. IEEE Computer Society, 1995. doi: 10.1109/FTCS.1995.466960.

[3] D. G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Appl. Math.*, 138(3):371–379, 2004. doi: 10.1016/j.dam.2003.07.001.

[4] D. G. Corneil, H. Kim, S. Natarajan, S. Olariu, and A. P. Sprague. Simple linear time recognition of unit interval graphs. *Inform. Process. Lett.*, 55(2):99–104, 1995. doi: 10.1016/0020-0190(95)00046-F.

[5] C. Crespelle. Fully dynamic representations of interval graphs. In *Graph-theoretic concepts in computer science*, vol. 5911 of *Lecture Notes in Comput. Sci.*, pp. 77–87. Springer, Berlin, 2010. doi: 10.1007/978-3-642-11409-0_7.

[6] C. Crespelle and C. Paul. Fully dynamic recognition algorithm and certificate for directed cographs. *Discrete Appl. Math.*, 154(12):1722–1741, 2006. doi: 10.1016/j.dam.2006.03.005.

[7] C. Crespelle and C. Paul. Fully dynamic algorithm for recognition and modular decomposition of permutation graphs. *Algorithmica*, 58(2):405–432, 2010. doi: 10.1007/s00453-008-9273-0.

[8] X. Deng, P. Hell, and J. Huang. Linear-time representation algorithms for proper circular-arc graphs and proper interval graphs. *SIAM J. Comput.*, 25(2):390–403, 1996. doi: 10.1137/S0097539792269095.

[9] E. Gioan and C. Paul. Split decomposition and graph-labelled trees: characterizations and fully dynamic algorithms for totally decomposable graphs. *Discrete Appl. Math.*, 160(6):708–733, 2012. doi: 10.1016/j.dam.2011.05.007.

[10] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*, vol. 57 of *Annals of Discrete Mathematics*. Elsevier Science B.V., Amsterdam, second edition, 2004.

[11] P. Heggernes and F. Mancini. Dynamically maintaining split graphs. *Discrete Appl. Math.*, 157(9): 2057–2069, 2009. doi: 10.1016/j.dam.2008.06.028.

[12] P. Hell and J. Huang. Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs. *SIAM J. Discrete Math.*, 18(3):554–570 (electronic), 2004/05. doi: 10.1137/S0895480103430259.

[13] P. Hell, R. Shamir, and R. Sharan. A fully dynamic algorithm for recognizing and representing proper interval graphs. *SIAM J. Comput.*, 31(1):289–305 (electronic), 2001. doi: 10.1137/S0097539700372216.

[14] C. M. Herrera de Figueiredo, J. Meidanis, and C. Picinin de Mello. A linear-time algorithm for proper interval graph recognition. *Inform. Process. Lett.*, 56(3):179–184, 1995. doi: 10.1016/0020-0190(95)00133-W.

[15] W. L. Hsu and K.-H. Tsai. Linear time algorithms on circular-arc graphs. *Inform. Process. Lett.*, 40(3):123–129, 1991.

[16] J. Huang. On the structure of local tournaments. *J. Combin. Theory Ser. B*, 63(2):200–221, 1995. doi: 10.1006/jctb.1995.1016.

[17] L. Ibarra. Fully dynamic algorithms for chordal graphs and split graphs. *ACM Trans. Algorithms*, 4(4):Art. 40, 20, 2008. doi: 10.1145/1383369.1383371.

[18] L. Ibarra. A fully dynamic graph algorithm for recognizing proper interval graphs. In *WALCOM—Algorithms and computation*, vol. 5431 of *Lecture Notes in Comput. Sci.*, pp. 190–201, Berlin, 2009. Springer. doi: 10.1007/978-3-642-00202-1_17.

[19] L. Ibarra. A fully dynamic graph algorithm for recognizing interval graphs. *Algorithmica*, 58(3): 637–678, 2010. doi: 10.1007/s00453-009-9291-6.

[20] H. Kaplan and Y. Nussbaum. Certifying algorithms for recognizing proper circular-arc graphs and unit circular-arc graphs. *Discrete Appl. Math.*, 157(15):3216–3230, 2009. doi: 10.1016/j.dam.2009.07.002.

[21] D. Kratsch, R. M. McConnell, K. Mehlhorn, and J. P. Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. *SIAM J. Comput.*, 36(2):326–353 (electronic), 2006.

[22] C. G. Lekkerkerker and J. C. Boland. Representation of a finite graph by a set of intervals on the real line. *Fund. Math.*, 51:45–64, 1962/1963.

[23] M. C. Lin and J. L. Szwarcfiter. Characterizations and recognition of circular-arc graphs and subclasses: a survey. *Discrete Math.*, 309(18):5618–5635, 2009. doi: 10.1016/j.disc.2008.04.003.

[24] M. C. Lin, F. J. Soulignac, and J. L. Szwarcfiter. Normal Helly circular-arc graphs and its subclasses. *Discrete Appl. Math.*, 161(7-8):1037–1059, 2013. doi: 10.1016/j.dam.2012.11.005.

[25] R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Comput. Sci. Rev.*, 5(2):119–161, 2011. doi: 10.1016/j.cosrev.2010.09.009.

[26] D. Meister. Recognition and computation of minimal triangulations for AT-free claw-free and co-comparability graphs. *Discrete Appl. Math.*, 146(3):193–218, 2005. doi: 10.1016/j.dam.2004.10.001.

[27] S. D. Nikolopoulos, L. Palios, and C. Papadopoulos. A fully dynamic algorithm for the recognition of P_4-sparse graphs. *Theoret. Comput. Sci.*, 439:41–57, 2012. doi: 10.1016/j.tcs.2012.03.020.

[28] M. Pirlot and P. Vincke. *Semiorders*, vol. 36 of *Theory and Decision Library. Series B: Mathematical and Statistical Methods*. Kluwer Academic Publishers Group, Dordrecht, 1997. doi: 10.1007/978-94-015-8883-6.

[29] R. Shamir and R. Sharan. A fully dynamic algorithm for modular decomposition and recognition of cographs. *Discrete Appl. Math.*, 136(2-3):329–340, 2004. doi: 10.1016/S0166-218X(03)00448-7.

[30] F. J. Soulignac. Fully dynamic recognition of proper circular-arc graphs. *Algorithmica*, 71(4): 904–968, 2015. doi: 10.1007/s00453-013-9835-7.

[31] M. Tedder and D. Corneil. An optimal, edges-only fully dynamic algorithm for distance-hereditary graphs. In *STACS 2007*, vol. 4393 of *Lecture Notes in Comput. Sci.*, pp. 344–355. Springer, Berlin, 2007. doi: 10.1007/978-3-540-70918-3_30.

[32] A. Tucker. Structure theorems for some circular-arc graphs. *Discrete Math.*, 7:167–195, 1974.

A Adequacy proofs

In this appendix we include the proofs that a family of semiblocks $\mathcal{B}$ is an adequate forbidden. These proofs were generated by a computer program, which explains why all the sections have the same structure. Each section references the lemma in which it is required. Then, a summary of the *current knowledge* of $H[\mathcal{B} \cup \{v\}]$ is depicted. By current knowledge we mean that the edges that we actually know that belong or could belong to $H[\mathcal{B}]$ are depicted. Together with this graph, we describe four fields:

- $\mathcal{B}$ contains all the semiblocks in $\mathcal{B}$ in the order in which they appear in the round representation.
- F_r shows the values of F_r but only for those semiblocks whose values of F_r cannot be deduced otherwise. Also, this is depicted taking into account only those edges that belong to $H[\mathcal{B}]$ using our current knowledge.
- $N(v)$ shows the semiblocks to which v is adjacent, according to our current knowledge.
- Rest includes all the adjacencies that could be added into $H[\mathcal{B}]$.

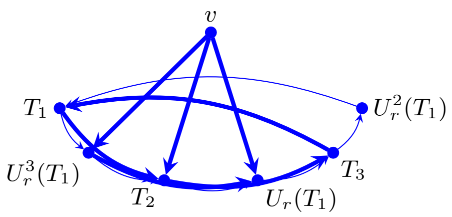
With this information, we can enumerate all the possible subgraphs that $\mathcal{B}$ could induce in H. Of course, there is only one such possibility for $H[\mathcal{B}]$ when Rest is empty; in such a case, this summary is not depicted (see e.g., Section A.1).

After the summary, there is one subsection dealing with each possibility for $H[\mathcal{B}]$, except for those that are duplicated. Each subsection includes the case stating "If ⟨case⟩, then ⟨forbidden subgraph⟩", where the forbidden subgraph is highlighted in blue. There are two kinds of duplicated possibilities: those in which $H[\mathcal{B}]$ is isomorphic to a case already examined, and those that are included in some other case. The former are ignored, while the latter are described in a section entitled "Implied cases".

A.1 Lemma 5.3 (a)

A.1.1

If $\emptyset$, then H_2



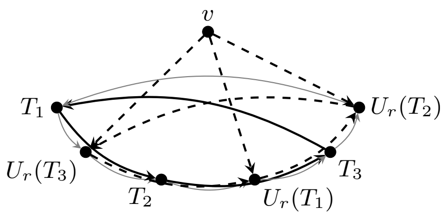
A.2 Lemma 5.3 (e)

$\mathcal{B}$: T_1, $U_r(T_3)$, T_2, $U_r(T_1)$, T_3, $U_r(T_2)$

F_r: $T_1 \longrightarrow T_2$, $T_2 \longrightarrow T_3$, $T_3 \longrightarrow T_1$

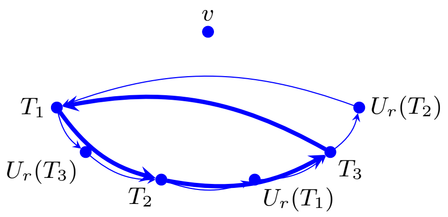
$N(v)$: $\emptyset$

Rest: $U_r(T_1) \longrightarrow U_r(T_2)$, $U_r(T_2) \longrightarrow U_r(T_3)$, $U_r(T_3) \longrightarrow U_r(T_1)$, $(v, U_r(T_3))$, $(v, U_r(T_2))$, $(v, U_r(T_1))$



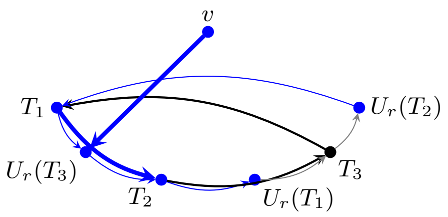
A.2.1

If $\emptyset$, then S_3^*



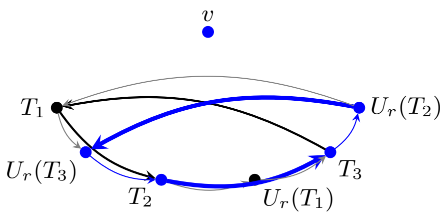
A.2.2

If $(v, U_r(T_3))$, then H_1



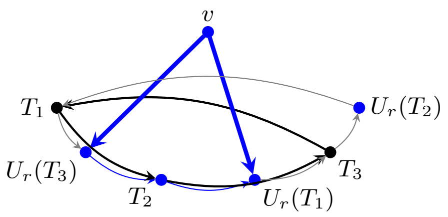
A.2.3

If $U_r(T_2) \longrightarrow U_r(T_3)$, then C_4^*



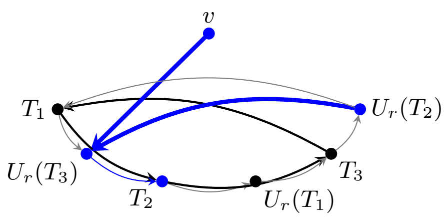
A.2.4

If $(v, U_r(T_3))$, $(v, U_r(T_1))$, then C_4^*



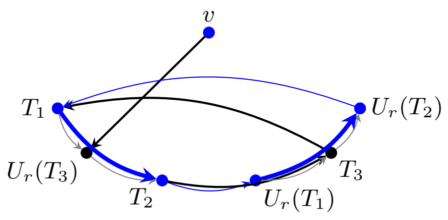
A.2.5

If $U_r(T_2) \longrightarrow U_r(T_3)$, $(v, U_r(T_3))$, then $K_{1,3}$



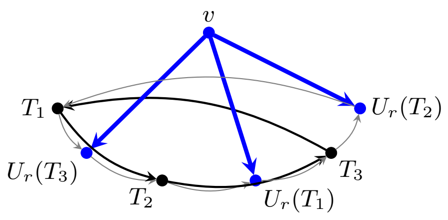
A.2.6

If $U_r(T_1) \longrightarrow U_r(T_2)$, $(v, U_r(T_3))$, then C_4^*



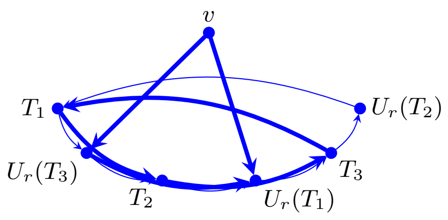
A.2.7

If $(v, U_r(T_3))$, $(v, U_r(T_1))$, $(v, U_r(T_2))$, then $K_{1,3}$



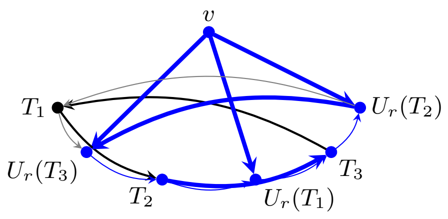
A.2.8

If $U_r(T_3) \longrightarrow U_r(T_1)$, $(v, U_r(T_3))$, $(v, U_r(T_1))$, then H_3



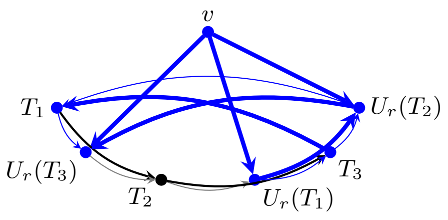
A.2.9

If $U_r(T_2) \longrightarrow U_r(T_3)$, $(v, U_r(T_3))$, $(v, U_r(T_1))$, $(v, U_r(T_2))$, then $\overline{C_6}$



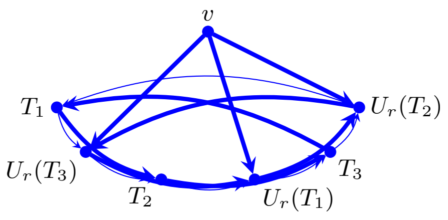
A.2.10

If $U_r(T_2) \longrightarrow U_r(T_3)$, $U_r(T_1) \longrightarrow U_r(T_2)$, $(v, U_r(T_3))$, $(v, U_r(T_1))$, $(v, U_r(T_2))$, then W_5



A.2.11

If $U_r(T_2) \longrightarrow U_r(T_3)$, $U_r(T_3) \longrightarrow U_r(T_1)$, $U_r(T_1) \longrightarrow U_r(T_2)$, $(v, U_r(T_3))$, $(v, U_r(T_1))$, $(v, U_r(T_2))$, then H_5



A.2.12 Implied cases

By A.2.3, at least one of $\{(v, U_r(T_3)), (v, U_r(T_2))\}$ must belong to the graph when all of $\{U_r(T_2) \longrightarrow U_r(T_3)\}$ are present. Thus, the following cases are solved:

12. $\{U_r(T_2) \longrightarrow U_r(T_3), U_r(T_1) \longrightarrow U_r(T_2)\}$,
13. $\{U_r(T_2) \longrightarrow U_r(T_3), U_r(T_3) \longrightarrow U_r(T_1), U_r(T_1) \longrightarrow U_r(T_2)\}$

By A.2.5, at least one of $\{(v, U_r(T_2))\}$ must belong to the graph when all of $\{U_r(T_2) \longrightarrow U_r(T_3), (v, U_r(T_3))\}$ are present. Thus, the following cases are solved:

14. $\{U_r(T_2) \longrightarrow U_r(T_3), (v, U_r(T_3)), (v, U_r(T_1))\}$,
15. $\{U_r(T_2) \longrightarrow U_r(T_3), U_r(T_1) \longrightarrow U_r(T_2), (v, U_r(T_3))\}$,
16. $\{U_r(T_2) \longrightarrow U_r(T_3), U_r(T_3) \longrightarrow U_r(T_1), (v, U_r(T_3))\}$,
17. $\{U_r(T_2) \longrightarrow U_r(T_3), U_r(T_1) \longrightarrow U_r(T_2), (v, U_r(T_3)), (v, U_r(T_1))\}$,
18. $\{U_r(T_2) \longrightarrow U_r(T_3), U_r(T_3) \longrightarrow U_r(T_1), (v, U_r(T_3)), (v, U_r(T_1))\}$,
19. $\{U_r(T_2) \longrightarrow U_r(T_3), U_r(T_3) \longrightarrow U_r(T_1), U_r(T_1) \longrightarrow U_r(T_2), (v, U_r(T_3))\}$,
20. $\{U_r(T_2) \longrightarrow U_r(T_3), U_r(T_3) \longrightarrow U_r(T_1), U_r(T_1) \longrightarrow U_r(T_2), (v, U_r(T_3)), (v, U_r(T_1))\}$

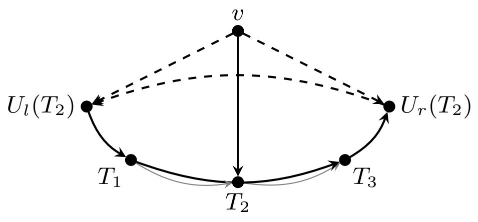
A.3 Lemma 5.3 (c)

$\mathcal{B}$: $U_l(T_2)$, T_1, T_2, T_3, $U_r(T_2)$

F_r: $T_3 \longrightarrow U_r(T_2)$, $U_l(T_2) \longrightarrow T_1$, $T_1 \longrightarrow T_3$

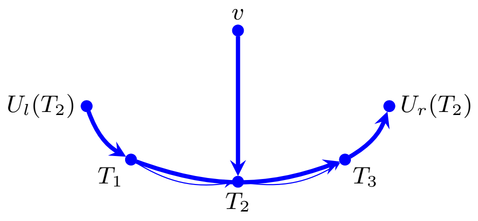
$N(v)$: T_2

Rest: $U_r(T_2) \longrightarrow U_l(T_2)$, $(v,\, U_r(T_2))$, $(v,\, U_l(T_2))$



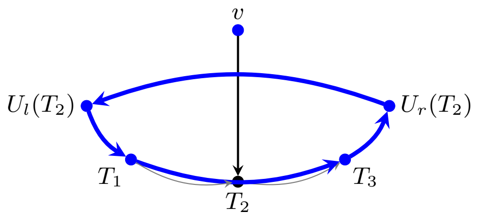
A.3.1

If $\emptyset$, then H_1



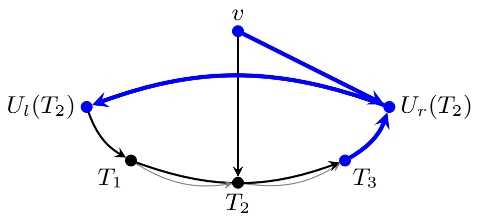
A.3.2

If $U_r(T_2) \longrightarrow U_l(T_2)$, then C_4^*



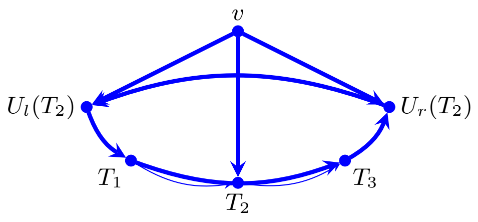
A.3.3

If $U_r(T_2) \longrightarrow U_l(T_2)$, $(v,\, U_r(T_2))$, then $K_{1,3}$



A.3.4

If $U_r(T_2) \longrightarrow U_l(T_2)$, $(v,\, U_l(T_2))$, $(v,\, U_r(T_2))$, then $\overline{C_6}$



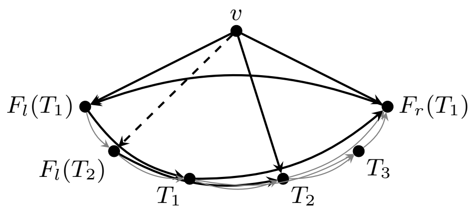
A.4 Lemma 5.3 (d)

$\mathcal{B}$: $F_l(T_1)$, $F_l(T_2)$, T_1, T_2, T_3, $F_r(T_1)$

F_r: $F_r(T_1) \longrightarrow F_l(T_1)$, $F_l(T_2) \longrightarrow T_2$, $F_l(T_1) \longrightarrow T_1$, $T_1 \longrightarrow F_r(T_1)$

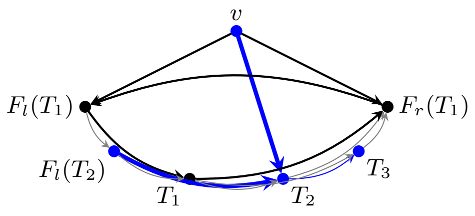
$N(v)$: $F_r(T_1)$, $F_l(T_1)$, T_2

Rest: $(v,\, F_l(T_2))$



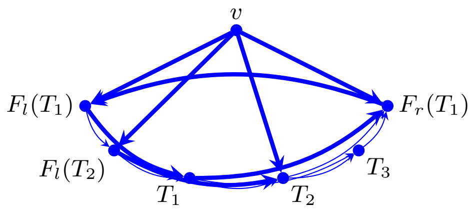
A.4.1

If $\emptyset$, then $K_{1,3}$



A.4.2

If $(v,\, F_l(T_2))$, then H_5



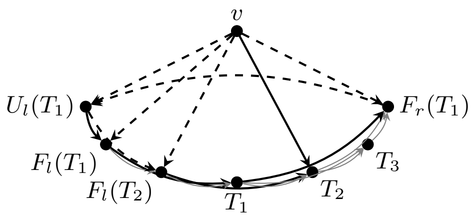
A.5 Lemma 5.3 (d)

$\mathcal{B}$: $U_l(T_1)$, $F_l(T_1)$, $F_l(T_2)$, T_1, T_2, T_3, $F_r(T_1)$

F_r: $U_l(T_1) \longrightarrow F_l(T_1)$, $F_l(T_2) \longrightarrow T_2$, $F_l(T_1) \longrightarrow T_1$, $T_1 \longrightarrow F_r(T_1)$

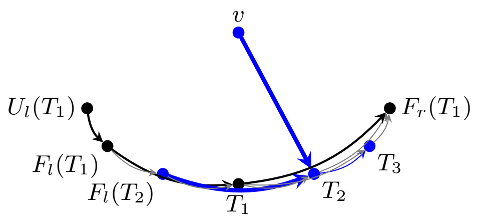
$N(v)$: T_2

Rest: $F_r(T_1) \longrightarrow U_l(T_1)$, $U_l(T_1) \longrightarrow F_l(T_2)$, $(v,\, U_l(T_1))$, $(v,\, F_r(T_1))$, $(v,\, F_l(T_1))$, $(v,\, F_l(T_2))$



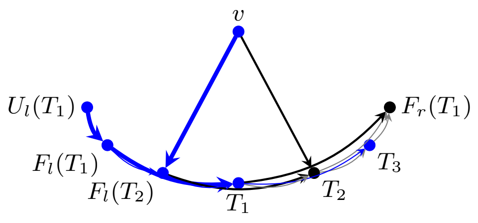
A.5.1

If $\emptyset$, then $K_{1,3}$



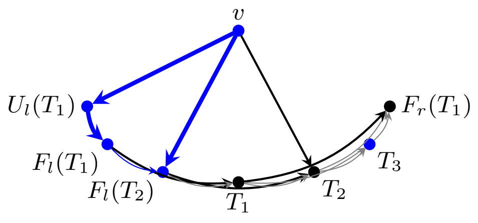
A.5.2

If $(v,\, F_l(T_2))$, then H_1



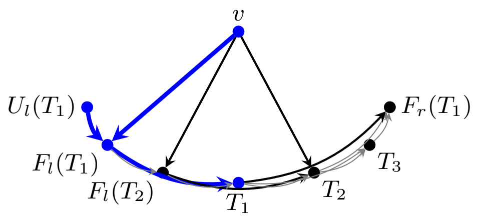
A.5.3

If $(v,\, F_l(T_2))$, $(v,\, U_l(T_1))$, then C_4^*



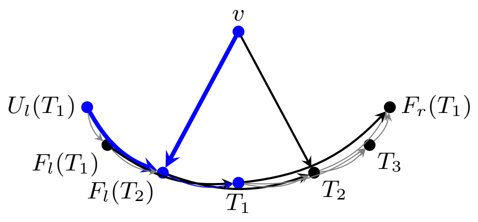
A.5.4

If $(v,\, F_l(T_2))$, $(v,\, F_l(T_1))$, then $K_{1,3}$



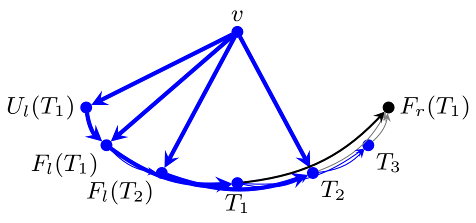
A.5.5

If $U_l(T_1) \longrightarrow F_l(T_2)$, $(v,\, F_l(T_2))$, then $K_{1,3}$



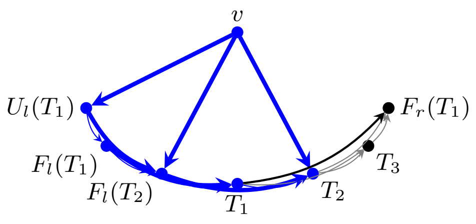
A.5.6

If $(v,\, F_l(T_2))$, $(v,\, U_l(T_1))$, $(v,\, F_l(T_1))$, then H_3



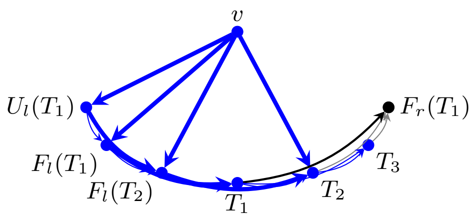
A.5.7

If $U_l(T_1) \longrightarrow F_l(T_2)$, $(v,\, F_l(T_2))$, $(v,\, U_l(T_1))$, then W_5



A.5.8

If $U_l(T_1) \longrightarrow F_l(T_2)$, $(v,\, F_l(T_2))$, $(v,\, U_l(T_1))$, $(v,\, F_l(T_1))$, then H_2



A.5.9 Implied cases

By A.5.1, at least one of $\{(v,\, F_l(T_2))\}$ must belong to the graph when all of $\{\emptyset\}$ are present. Thus, the following cases are solved:

9. $\{(v,\, F_r(T_1))\}$,
10. $\{F_r(T_1) \longrightarrow U_l(T_1)\}$,
11. $\{(v,\, U_l(T_1))\}$,
12. $\{(v,\, F_l(T_1))\}$,
13. $\{U_l(T_1) \longrightarrow F_l(T_2)\}$,
14. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_r(T_1))\}$,
15. $\{(v,\, F_r(T_1)),\, (v,\, U_l(T_1))\}$,
16. $\{(v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$,
17. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_r(T_1))\}$,
18. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, U_l(T_1))\}$,
19. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_l(T_1))\}$,
20. $\{F_r(T_1) \longrightarrow U_l(T_1),\, U_l(T_1) \longrightarrow F_l(T_2)\}$,
21. $\{(v,\, U_l(T_1)),\, (v,\, F_l(T_1))\}$,
22. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, U_l(T_1))\}$,

23. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_1))\}$,

24. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_r(T_1)),\, (v,\, U_l(T_1))\}$,

25. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$,

26. $\{F_r(T_1) \longrightarrow U_l(T_1),\, U_l(T_1) \longrightarrow F_l(T_2),$ $(v,\, F_r(T_1))\}$,

27. $\{(v,\, F_r(T_1)),\, (v,\, U_l(T_1)),\, (v,\, F_l(T_1))\}$,

28. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_r(T_1)),\, (v,\, U_l(T_1))\}$,

29. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$,

30. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, U_l(T_1)),\, (v,\, F_l(T_1))\}$,

31. $\{F_r(T_1) \longrightarrow U_l(T_1),\, U_l(T_1) \longrightarrow F_l(T_2),$ $(v,\, U_l(T_1))\}$,

32. $\{F_r(T_1) \longrightarrow U_l(T_1),\, U_l(T_1) \longrightarrow F_l(T_2),$ $(v,\, F_l(T_1))\}$,

33. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, U_l(T_1)),\, (v,\, F_l(T_1))\}$,

34. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, U_l(T_1)),\, (v,\, F_r(T_1)),$ $(v,\, F_l(T_1))\}$,

35. $\{U_l(T_1) \longrightarrow F_l(T_2),\, F_r(T_1) \longrightarrow U_l(T_1),$ $(v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$,

36. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, U_l(T_1)),\, (v,\, F_r(T_1)),$ $(v,\, F_l(T_1))\}$,

37. $\{U_l(T_1) \longrightarrow F_l(T_2),\, F_r(T_1) \longrightarrow U_l(T_1),$ $(v,\, U_l(T_1)),\, (v,\, F_l(T_1))\}$,

38. $\{U_l(T_1) \longrightarrow F_l(T_2),\, F_r(T_1) \longrightarrow U_l(T_1),$ $(v,\, U_l(T_1)),\, (v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$

By A.5.2, at least one of $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, U_l(T_1)),$ $(v,\, F_l(T_1))\}$ must belong to the graph when all of $\{(v,\, F_l(T_2))\}$ are present. Thus, the following cases are solved:

39. $\{(v,\, F_l(T_2)),\, (v,\, F_r(T_1))\}$,

40. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_l(T_2))\}$,

41. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_l(T_2)),\, (v,\, F_r(T_1))\}$

By A.5.3, at least one of $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_1))\}$ must belong to the graph when all of $\{(v,\, F_l(T_2)),$ $(v,\, U_l(T_1))\}$ are present. Thus, the following cases are solved:

42. $\{(v,\, F_l(T_2)),\, (v,\, F_r(T_1)),\, (v,\, U_l(T_1))\}$,

43. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_l(T_2)),\, (v,\, U_l(T_1))\}$,

44. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_l(T_2)),\, (v,\, U_l(T_1)),$ $(v,\, F_r(T_1))\}$

By A.5.4, at least one of $\{(v,\, U_l(T_1))\}$ must belong to the graph when all of $\{(v,\, F_l(T_2)),\, (v,\, F_l(T_1))\}$ are present. Thus, the following cases are solved:

45. $\{(v,\, F_l(T_2)),\, (v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$,

46. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_l(T_2)),\, (v,\, F_l(T_1))\}$,

47. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_2)),\, (v,\, F_l(T_1))\}$,

48. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_l(T_2)),\, (v,\, F_r(T_1)),$ $(v,\, F_l(T_1))\}$,

49. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_2)),\, (v,\, F_r(T_1)),$ $(v,\, F_l(T_1))\}$,

50. $\{U_l(T_1) \longrightarrow F_l(T_2),\, F_r(T_1) \longrightarrow U_l(T_1),$ $(v,\, F_l(T_2)),\, (v,\, F_l(T_1))\}$,

51. $\{U_l(T_1) \longrightarrow F_l(T_2),\, F_r(T_1) \longrightarrow U_l(T_1),$ $(v,\, F_l(T_2)),\, (v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$

By A.5.5, at least one of $\{(v,\, U_l(T_1))\}$ must belong to the graph when all of $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_2))\}$ are present. Thus, the following cases are solved:

52. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_2)),\, (v,\, F_r(T_1))\}$,

53. $\{F_r(T_1) \longrightarrow U_l(T_1),\, U_l(T_1) \longrightarrow F_l(T_2),$ $(v,\, F_l(T_2))\}$,

54. $\{U_l(T_1) \longrightarrow F_l(T_2),\, F_r(T_1) \longrightarrow U_l(T_1),$ $(v,\, F_l(T_2)),\, (v,\, F_r(T_1))\}$

By A.5.6, at least one of $\{U_l(T_1) \longrightarrow F_l(T_2)\}$ must belong to the graph when all of $\{(v,\, F_l(T_2)),\, (v,\, U_l(T_1)),$ $(v,\, F_l(T_1))\}$ are present. Thus, the following cases are solved:

55. $\{(v,\, F_l(T_2)),\, (v,\, U_l(T_1)),\, (v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$,

56. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_l(T_2)),\, (v,\, U_l(T_1)),$ $(v,\, F_l(T_1))\}$,

57. $\{F_r(T_1) \longrightarrow U_l(T_1),\, (v,\, F_l(T_2)),\, (v,\, U_l(T_1)),$ $(v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$

By A.5.7, at least one of $\{(v,\, F_l(T_1))\}$ must belong to the graph when all of $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_2)),$ $(v,\, U_l(T_1))\}$ are present. Thus, the following cases are solved:

58. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_2)),\, (v,\, U_l(T_1)),$ $(v,\, F_r(T_1))\}$,

59. $\{U_l(T_1) \longrightarrow F_l(T_2),\, F_r(T_1) \longrightarrow U_l(T_1),$ $(v,\, F_l(T_2)),\, (v,\, U_l(T_1))\}$

By A.5.8, at least one of $\{\emptyset\}$ must belong to the graph when all of $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_2)),\, (v,\, U_l(T_1)),$ $(v,\, F_l(T_1))\}$ are present. Thus, the following cases are solved:

60. $\{U_l(T_1) \longrightarrow F_l(T_2),\, (v,\, F_l(T_2)),\, (v,\, U_l(T_1)),$ $(v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$,

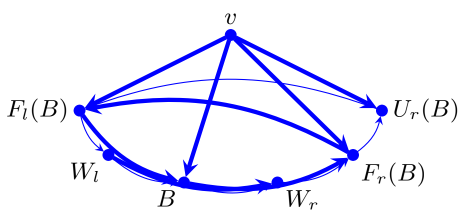
61. $\{U_l(T_1) \longrightarrow F_l(T_2),\, F_r(T_1) \longrightarrow U_l(T_1),$ $(v,\, F_l(T_2)),\, (v,\, U_l(T_1)),\, (v,\, F_l(T_1))\}$,

62. $\{U_l(T_1) \longrightarrow F_l(T_2),\, F_r(T_1) \longrightarrow U_l(T_1),$ $(v,\, F_l(T_2)),\, (v,\, U_l(T_1)),\, (v,\, F_r(T_1)),\, (v,\, F_l(T_1))\}$

A.6 Lemma 5.4

A.6.1

If $\emptyset$, then H_4



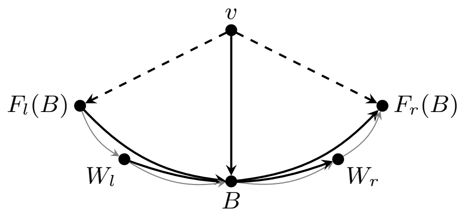
A.7 Lemma 5.5

$\mathcal{B}$: $F_l(B)$, W_l, B, W_r, $F_r(B)$

F_r: $B \longrightarrow F_r(B)$, $W_l \longrightarrow W_r$, $F_l(B) \longrightarrow B$

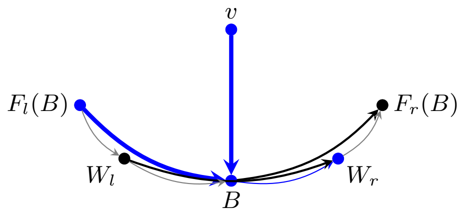
$N(v)$: B

Rest: $(v, F_l(B))$, $(v, F_r(B))$



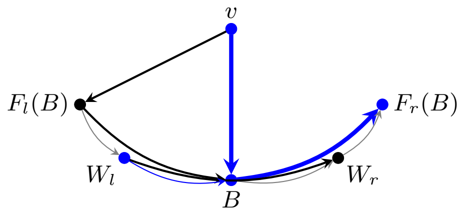
A.7.1

If $\emptyset$, then $K_{1,3}$



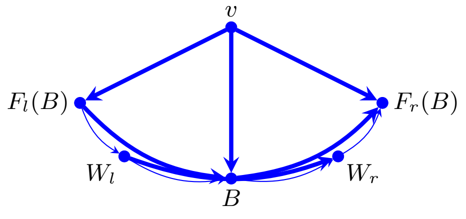
A.7.2

If $(v, F_l(B))$, then $K_{1,3}$



A.7.3

If $(v, F_r(B))$, $(v, F_l(B))$, then W_5



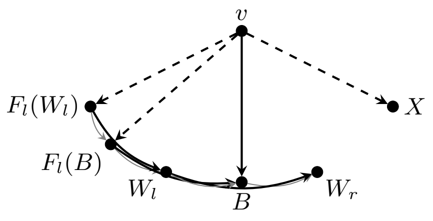
A.8 Lemma 5.5

$\mathcal{B}$: $F_l(W_l)$, $F_l(B)$, W_l, B, W_r, X

F_r: $F_l(W_l) \longrightarrow W_l$, $W_l \longrightarrow W_r$, $F_l(B) \longrightarrow B$

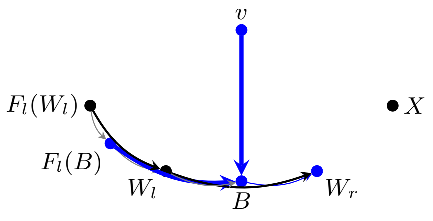
$N(v)$: B

Rest: $(v, F_l(W_l))$, (v, X), $(v, F_l(B))$



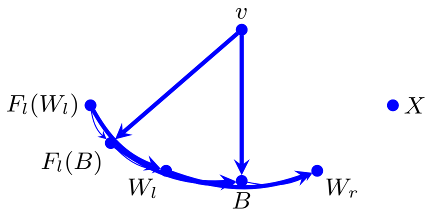
A.8.1

If $\emptyset$, then $K_{1,3}$



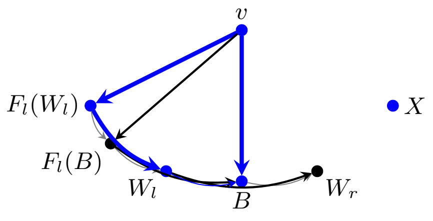
A.8.2

If $(v, F_l(B))$, then S_3^*



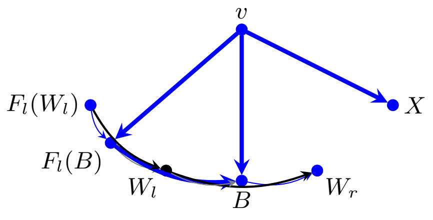
A.8.3

If $(v, F_l(W_l))$, $(v, F_l(B))$, then C_4^*



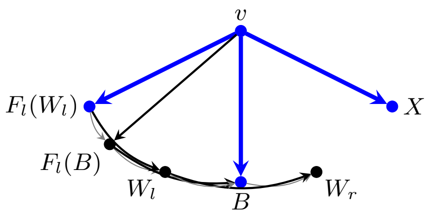
A.8.4

If (v, X), $(v, F_l(B))$, then H_1



A.8.5

If $(v,\, X)$, $(v,\, F_l(W_l))$, $(v,\, F_l(B))$, then $K_{1,3}$



A.8.6 Implied cases

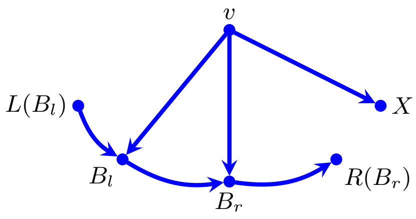
By A.8.1, at least one of $\{(v,\, F_l(B))\}$ must belong to the graph when all of $\{\emptyset\}$ are present. Thus, the following cases are solved:

6. $\{(v,\, F_l(W_l))\}$,
7. $\{(v,\, X)\}$,
8. $\{(v,\, X),\, (v,\, F_l(W_l))\}$

A.9 Lemma 5.7

A.9.1

If $\emptyset$, then H_1



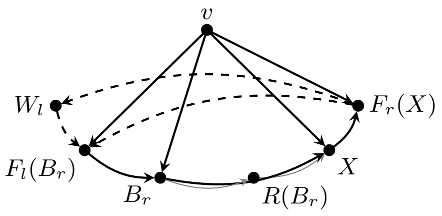
A.10 Lemma 5.7

$\mathcal{B}$: W_l, $F_l(B_r)$, B_r, $R(B_r)$, X, $F_r(X)$

F_r: $B_r \longrightarrow X$, $X \longrightarrow F_r(X)$, $F_l(B_r) \longrightarrow B_r$

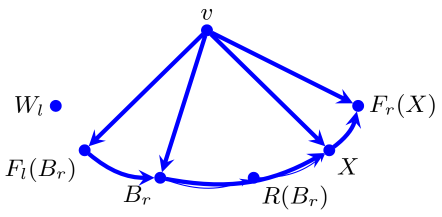
$N(v)$: $F_r(X)$, X, $F_l(B_r)$, B_r

Rest: $F_r(X) \longrightarrow F_l(B_r)$, $F_r(X) \longrightarrow W_l$,
$W_l \longrightarrow F_l(B_r)$



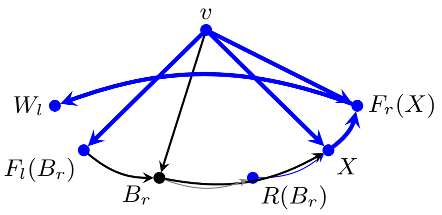
A.10.1

If $\emptyset$, then S_3^*



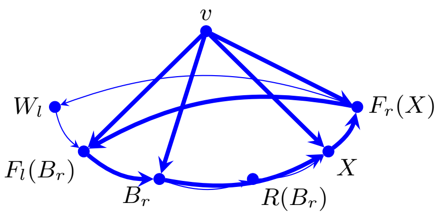
A.10.2

If $F_r(X) \longrightarrow W_l$, then H_1



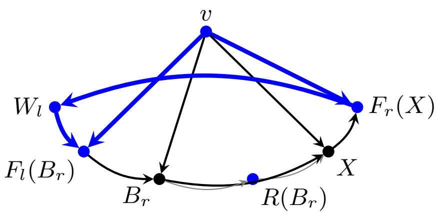
A.10.3

If $F_r(X) \longrightarrow F_l(B_r)$, $W_l \longrightarrow F_l(B_r)$, $F_r(X) \longrightarrow W_l$, then H_3



A.10.4

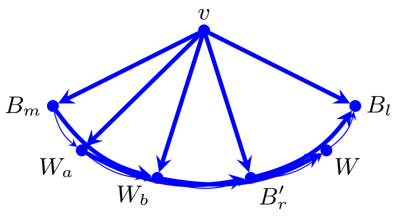
If $W_l \longrightarrow F_l(B_r)$, $F_r(X) \longrightarrow W_l$, then C_4^*



A.11 Lemma 5.10 (4)

A.11.1

If $\emptyset$, then H_4



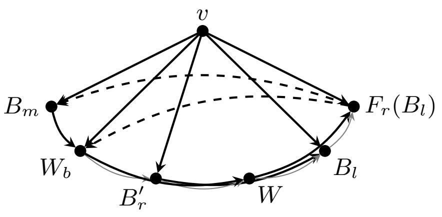
A.12 Lemma 5.10 (4)

$\mathcal{B}$: B_m, W_b, B_r, W, B_l, $F_r(B_l)$

F_r: $B_r \longrightarrow B_l$, $W \longrightarrow F_r(B_l)$, $W_b \longrightarrow W$, $B_m \longrightarrow W_b$

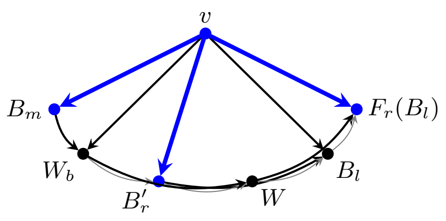
$N(v)$: B_r, B_l, $F_r(B_l)$, W_b, B_m

Rest: $F_r(B_l) \longrightarrow B_m$, $F_r(B_l) \longrightarrow W_b$



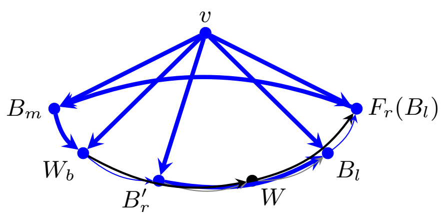
A.12.1

If $\emptyset$, then $K_{1,3}$



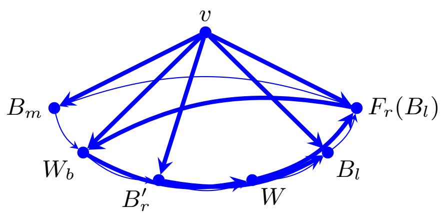
A.12.2

If $F_r(B_l) \longrightarrow B_m$, then W_5



A.12.3

If $F_r(B_l) \longrightarrow W_b$, $F_r(B_l) \longrightarrow B_m$, then H_5



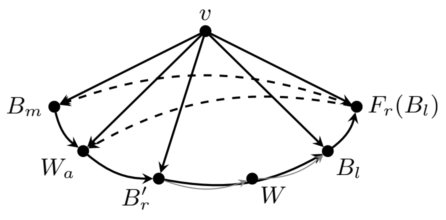
A.13 Lemma 5.10 (4)

$\mathcal{B}$: B_m, W_a, B_r, W, B_l, $F_r(B_l)$

F_r: $B_r \longrightarrow B_l$, $B_m \longrightarrow W_a$, $W_a \longrightarrow B_r$, $B_l \longrightarrow F_r(B_l)$

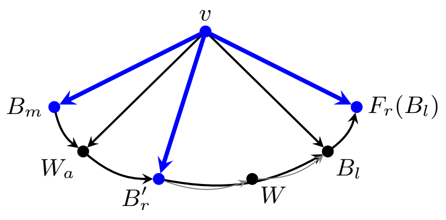
$N(v)$: B_r, W_a, $F_r(B_l)$, B_l, B_m

Rest: $F_r(B_l) \longrightarrow W_a$, $F_r(B_l) \longrightarrow B_m$



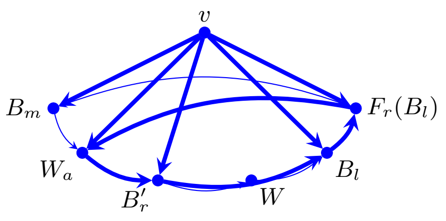
A.13.1

If $\emptyset$, then $K_{1,3}$



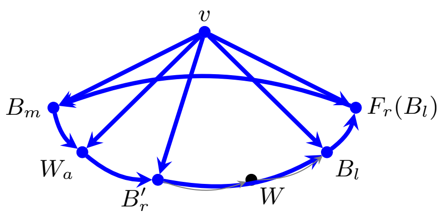
A.13.2

If $F_r(B_l) \longrightarrow B_m$, $F_r(B_l) \longrightarrow W_a$, then H_2



A.13.3

If $F_r(B_l) \longrightarrow B_m$, then W_5



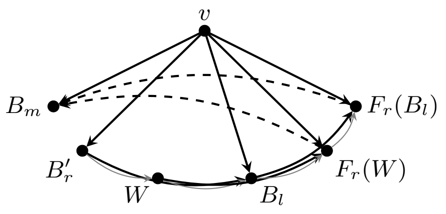
A.14 Lemma 5.10 (4)

$\mathcal{B}$: B_m, B_r, W, B_l, $F_r(W)$, $F_r(B_l)$

F_r: $B_r \longrightarrow B_l$, $B_l \longrightarrow F_r(B_l)$, $W \longrightarrow F_r(W)$

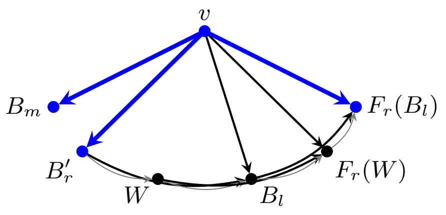
$N(v)$: B_r, $F_r(W)$, B_l, $F_r(B_l)$, B_m

Rest: $F_r(W) \longrightarrow B_m$, $F_r(B_l) \longrightarrow B_m$



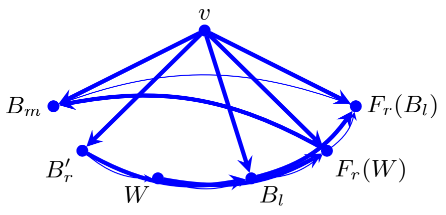
A.14.1

If $\emptyset$, then $K_{1,3}$



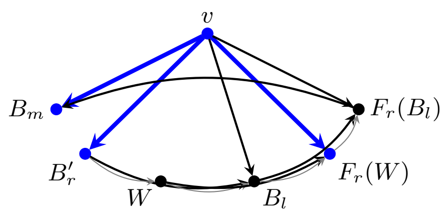
A.14.2

If $F_r(W) \longrightarrow B_m$, $F_r(B_l) \longrightarrow B_m$, then H_4



A.14.3

If $F_r(B_l) \longrightarrow B_m$, then $K_{1,3}$



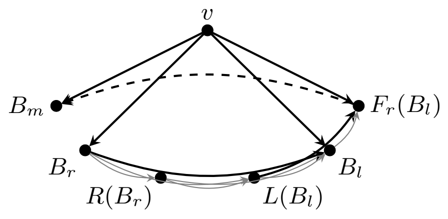
A.15 Lemma 5.10

$\mathcal{B}$: B_m, B_r, $R(B_r)$, $L(B_l)$, B_l, $F_r(B_l)$

F_r: $B_r \longrightarrow B_l$, $L(B_l) \longrightarrow F_r(B_l)$

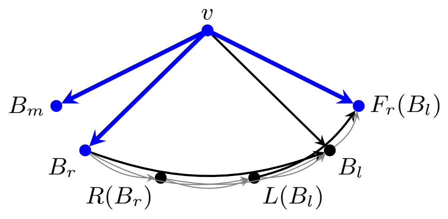
$N(v)$: B_l, $F_r(B_l)$, B_r, B_m

Rest: $F_r(B_l) \longrightarrow B_m$



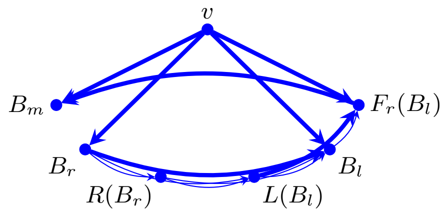
A.15.1

If $\emptyset$, then $K_{1,3}$



A.15.2

If $F_r(B_l) \longrightarrow B_m$, then H_2



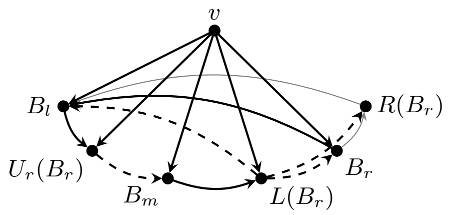
A.16 Lemma 5.10

$\mathcal{B}$: B_l, $U_r(B_r)$, B_m, $L(B_r)$, B_r, $R(B_r)$

F_r: $B_l \longrightarrow U_r(B_r)$, $B_r \longrightarrow B_l$, $B_m \longrightarrow L(B_r)$

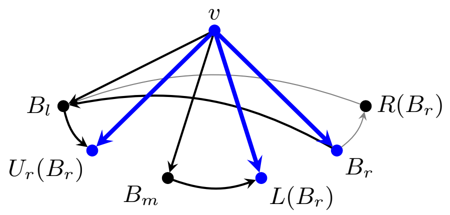
$N(v)$: $L(B_r)$, B_r, B_l, $U_r(B_r)$, B_m

Rest: $L(B_r) \longrightarrow B_l$, $L(B_r) \longrightarrow B_r$, $U_r(B_r) \longrightarrow B_m$, $L(B_r) \longrightarrow R(B_r)$



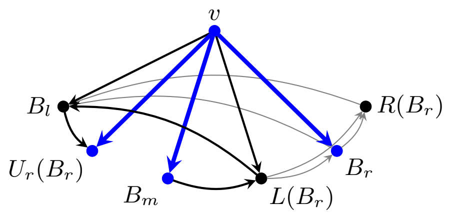
A.16.1

If $\emptyset$, then $K_{1,3}$



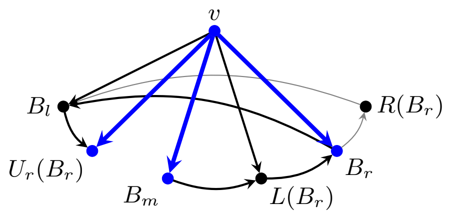
A.16.2

If $L(B_r) \longrightarrow B_l$, $L(B_r) \longrightarrow R(B_r)$, $L(B_r) \longrightarrow B_r$, then $K_{1,3}$



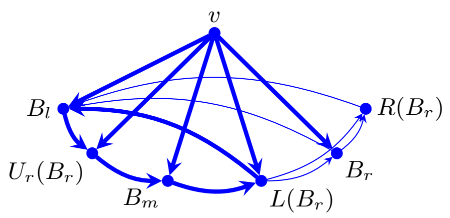
A.16.3

If $L(B_r) \longrightarrow B_r$, then $K_{1,3}$



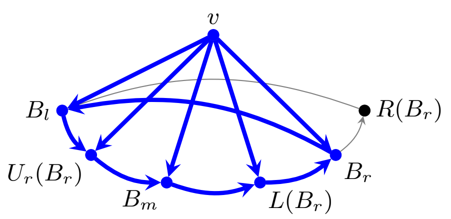
A.16.4

If $U_r(B_r) \longrightarrow B_m$, $L(B_r) \longrightarrow B_l$, $L(B_r) \longrightarrow R(B_r)$, $L(B_r) \longrightarrow B_r$, then H_4



A.16.5

If $U_r(B_r) \longrightarrow B_m$, $L(B_r) \longrightarrow B_r$, then W_5



A.16.6 Implied cases

By A.16.1, at least one of $\{L(B_r) \longrightarrow B_r\}$ must belong to the graph when all of $\{\emptyset\}$ are present. Thus, the following cases are solved:

6. $\{U_r(B_r) \longrightarrow B_m\}$

By A.16.3, at least one of $\{U_r(B_r) \longrightarrow B_m\}$ must belong to the graph when all of $\{L(B_r) \longrightarrow B_r\}$ are present. Thus, the following cases are solved:

7. $\{L(B_r) \longrightarrow R(B_r),\ L(B_r) \longrightarrow B_r\}$

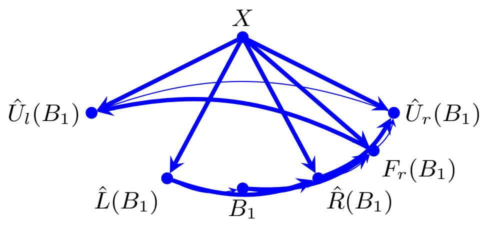
By A.16.5, at least one of $\{L(B_r) \longrightarrow B_l\}$ must belong to the graph when all of $\{U_r(B_r) \longrightarrow B_m,\ L(B_r) \longrightarrow B_r\}$ are present. Thus, the following cases are solved:

8. $\{U_r(B_r) \longrightarrow B_m,\ L(B_r) \longrightarrow R(B_r),$
   $L(B_r) \longrightarrow B_r\}$

A.17 Lemma 5.14

A.17.1

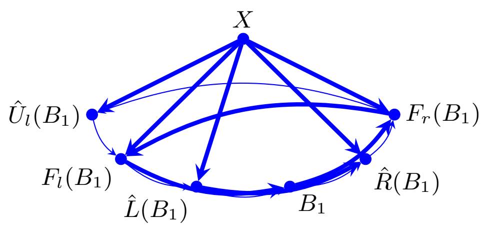
If $\emptyset$, then H_4



A.18 Lemma 5.14

A.18.1

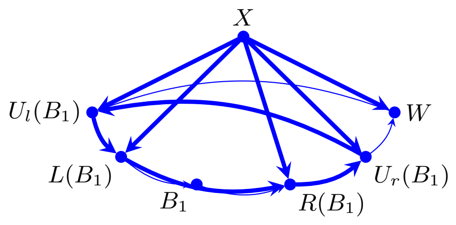
If $\emptyset$, then H_5



A.19 Lemma 5.14

A.19.1

If $\emptyset$, then H_2



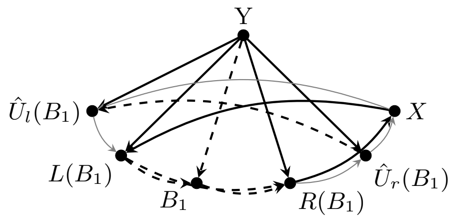
A.20 Lemma 5.14

$\mathcal{B}$: $\hat{U}_l(B_1)$, $L(B_1)$, B_1, $R(B_1)$, $\hat{U}_r(B_1)$, X

F_r: $R(B_1) \longrightarrow X$, $X \longrightarrow L(B_1)$

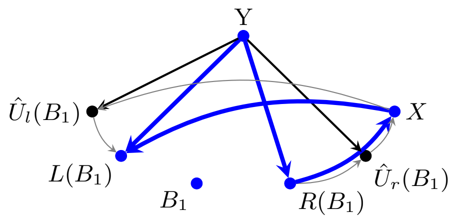
$N(Y)$: $L(B_1)$, $\hat{U}_l(B_1)$, $\hat{U}_r(B_1)$, $R(B_1)$

Rest: $\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)$, $B_1 \longrightarrow R(B_1)$,
$L(B_1) \longrightarrow B_1$, $L(B_1) \longrightarrow R(B_1)$, (Y, B_1)



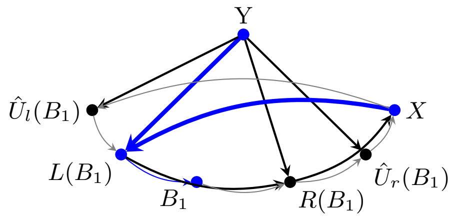
A.20.1

If $\emptyset$, then C_4^*



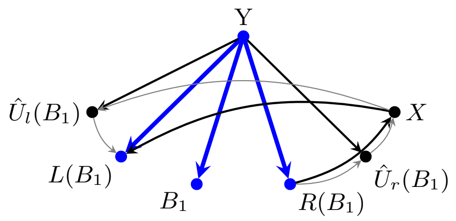
A.20.2

If $L(B_1) \longrightarrow R(B_1)$, $B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1$, then $K_{1,3}$



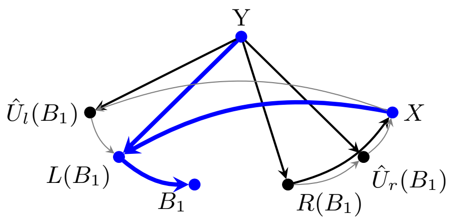
A.20.3

If (Y, B_1), then $K_{1,3}$



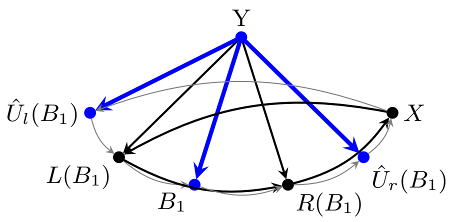
A.20.4

If $L(B_1) \longrightarrow B_1$, then $K_{1,3}$



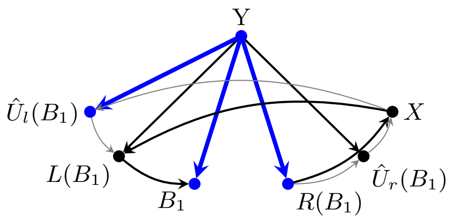
A.20.5

If $L(B_1) \longrightarrow R(B_1)$, $B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1$, (Y, B_1), then $K_{1,3}$



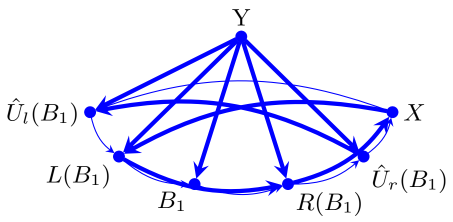
A.20.6

If $L(B_1) \longrightarrow B_1$, (Y, B_1), then $K_{1,3}$



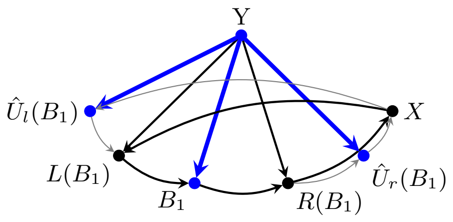
A.20.7

If $\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)$, $L(B_1) \longrightarrow R(B_1)$, $B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1$, (Y, B_1), then H_5



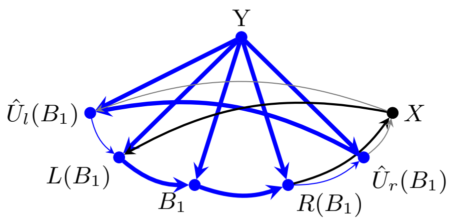
A.20.8

If $B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1$, (Y, B_1), then $K_{1,3}$



A.20.9

If $\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)$, $B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1$, (Y, B_1), then W_5



A.20.10 Implied cases

By A.20.1, at least one of $\{B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1$, $L(B_1) \longrightarrow R(B_1)$, (Y, B_1)$\}$ must belong to the graph when all of $\{\emptyset\}$ are present. Thus, the following cases are solved:

10. $\{\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)\}$

By A.20.2, at least one of $\{$(Y, B_1)$\}$ must belong to the graph when all of $\{L(B_1) \longrightarrow R(B_1)$, $B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1\}$ are present. Thus, the following cases are solved:

11. $\{\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)$, $L(B_1) \longrightarrow R(B_1)$, $B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1\}$

By A.20.3, at least one of $\{B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1$, $L(B_1) \longrightarrow R(B_1)\}$ must belong to the graph when all of $\{$(Y, B_1)$\}$ are present. Thus, the following cases are solved:

12. $\{\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)$, (Y, B_1)$\}$

By A.20.4, at least one of $\{$(Y, B_1)$\}$ must belong to the graph when all of $\{L(B_1) \longrightarrow B_1\}$ are present. Thus, the following cases are solved:

13. $\{\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)$, $L(B_1) \longrightarrow B_1\}$,
14. $\{B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1\}$,
15. $\{\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)$, $B_1 \longrightarrow R(B_1)$, $L(B_1) \longrightarrow B_1\}$

By A.20.6, at least one of $\{B_1 \longrightarrow R(B_1)\}$ must belong to the graph when all of $\{L(B_1) \longrightarrow B_1$, (Y, B_1)$\}$ are present. Thus, the following cases are solved:

16. $\{\hat{U}_r(B_1) \longrightarrow \hat{U}_l(B_1)$, $L(B_1) \longrightarrow B_1$, (Y, B_1)$\}$