Circle Graphs Can Be Recognized in Linear Time

Christophe Paul
CNRS, Université Montpellier, France

Ignaz Rutter
University of Passau, Germany

Abstract

To date, the best circle graph recognition algorithm, due to Gioan *et al.* [19] runs in almost linear time as it relies on a split decomposition algorithm [20] that uses the union-find data-structure [16, 34]. We show that in the case of circle graphs, the PC-tree data-structure [31] allows one to avoid the union-find data-structure to compute the split decomposition in linear time. As a consequence, we obtain the first linear-time recognition algorithm for circle graphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Graph theory; Theory of computation → Data structures design and analysis; Theory of computation → Graph algorithms analysis

Keywords and phrases graph classes, circle graphs, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.72

Related Version *Full Version*: `https://arxiv.org/abs/2512.23492`

Funding *Christophe Paul*: Research supported by ANR-DFG project GODASse ANR-24-CE48-4377.

1 Introduction

A *circle graph* is the intersection graph of a set of chords inscribed in a circle, called *chord diagram* (see Figure 1 below for an example).

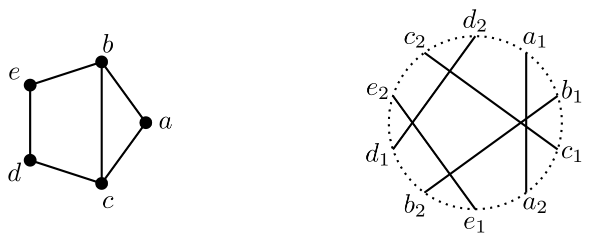

Figure 1 The House graph on the left and its chord diagram on the right.

Since their introduction in the early 70's, circle graphs have been extensively studied. They were first defined by Even and Itai [12], who proved that the minimum number of parallel stacks needed to sort a permutation equals the chromatic number of a circle graph. Independently, Bouchet [1] studied *alternance graphs* to provide an algorithmic solution of the Gauss problem on self-intersection curves in the plane. A double occurrence word on a set L of letters is a word containing exactly two copies of every letter of L. An alternance graph is defined by a double occurrence word on its set of vertices such that two vertices are adjacent if and only if their occurrences alternate in the word (see Subsection 3.1). It is easy to see that alternance graphs are exactly circle graphs. A first characterization of circle graphs was proposed by Fournier [14] in terms of ordered sets, while De Fraysseix [11] characterized circle graphs as intersection graphs of co-cyclic paths. As confirmed by recent results, circle graphs

43rd International Symposium on Theoretical Aspects of Computer Science (STACS 2026).
Editors: Meena Mahajan, Florin Manea, Annabelle McIver, and Nguyễn Kim Thắng
Article No. 72; pp. 72:1–72:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

play a very important role in the theory of *vertex minors* (see [27]). A *local-complementation* consists in replacing in a graph the neighborhood of a vertex by its complement graph. A vertex-minor of a graph G is a graph H that can be obtained from G by a series of local complementations and vertex deletions. It is easy to see from a chord diagram that every vertex-minor of a circle graph is itself a vertex minor (this was first noticed by Kotzig [24] in a series of seminars). Bouchet [3] showed that a graph is a circle graph if and only if it excludes one of the three graphs of Figure 2 as a vertex-minor. It is conjectured that the vertex-minor relation forms a well-quasi-ordering (see [27]). To tackle this question, the *rank-width* parameter has been introduced as the analog, for vertex-minor relation, of the tree-width for the minor relation [29]. It turns out that circle graphs play the same role for the rank-width parameter, as planar graphs for treewidth. Indeed Geelen *et al.* [17] recently proved an analog of the grid minor exclusion theorem [28]: a graph has bounded rank-width if and only if it excludes as a vertex minor a sufficiently large comparability grid[1]. In the same way as every planar graph is a minor of a large enough grid, every circle graph is a vertex minor of a sufficiently large comparability grid.

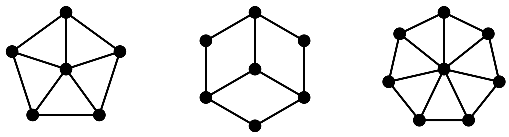

Figure 2 The vertex minor obstructions of circle graphs: W_5, the wheel of 5 vertices, F_7 and W_7.

As pointed out in [2], neither Fournier's nor de Fraysseix's characterization yields a polynomial-time recognition algorithm for circle graph. It took about 15 years for the first recognition algorithm to appear. Actually, three polynomial-time recognition algorithms were independently obtained in the mid 80's by Naji [26], by Gabor *et al.* [15], and by Bouchet [2], respectively. Each of these three algorithms involves the *split decomposition* of graphs, introduced by Cunningham and Edmonds [8]. Indeed, they all reduce the circle graph recognition problem to *prime* graphs, that is, graphs that are indecomposable with respect to the split decomposition. The $O(n^7)$-time algorithm of Naji [26] relies on the resolution of a system of linear equations that characterizes prime circle graphs. Bouchet's algorithm [2] runs in $O(n^5)$-time and uses the property that every prime circle graph on n vertices contains as a vertex minor a prime circle graph on $n-1$ vertices. The complexity of the algorithm of Gabor *et al.* is $O(n^3)$. It relies on the property that prime circle graphs have a unique chord diagram, a property that was formally proved only later (see [2, 7, 19]). The complexity of the circle graph recognition was later improved down to $O(n^2)$-time by Spinrad [33]. This was possible by the use of a novel $O(n^2)$-time algorithm to compute the split decomposition of a graph due to Ma and Spinrad [25]. We observe that the linear-time split decomposition algorithm later obtained by Dahlhaus [9, 10] has no impact on the complexity of Spinrad's circle graph recognition algorithm. The reason is that Spinrad's algorithm is a vertex-incremental algorithm, while Dahlhaus' is not. Finally, until very recently no subquadratic time circle graph recognition algorithm was known. Gioan *et al.* [20, 19] broke the quadratic barrier by designing a novel almost linear-time split decomposition algorithm and showed how to apply that algorithm for the circle graph recognition problem. Their algorithm uses an important

[1] For a positive integer n, the $n \times n$-comparability grid is the graph with vertex set $\{(i,j) \mid i,j \in \{1,2,...,n\}\}$ such that vertices (i,j) and (i',j') are adjacent if either $i \leq i'$ and $j \leq j'$, or $i \geq i'$ and $j \geq j'$.

property of the last vertex of a Lexicographic Breadth-First-Search (LexBFS) [30] with respect to the split-decomposition. In the case of circle graphs that property translates to the existence of a chord diagram where the neighborhood of the last LexBFS vertex appears *consecutively* (see Lemma 12). This allows Gioan *et al.* to design a LexBFS-based algorithm that incrementally updates the split decomposition and the corresponding representation of circle graphs. The hurdle to linear time complexity in the algorithm of Gioan *et al.* is the use of the *union-find* data-structure [16, 34] to update the parent-children relationships in the modified split decomposition. The resulting time complexity is $O((n+m)\cdot\alpha(n,m))$, where $\alpha(\cdot)$ is the slowly growing inverse of the Ackermann function.

Our result. Gioan *et al.* [19] left open the question of the existence of a linear-time circle graph recognition algorithm. To resolve that question, we show that the consecutive property of the neighborhood of the last vertex of a LexBFS ordering allows us to use a PC-tree data-structure [31] (see also [21, 13]) rather than the union-find data-structure. As a consequence, we shave the $\alpha(n,m)$ factor in the time complexity of Gioan *et al.*'s algorithm, yielding the first linear-time circle graph recognition algorithm.

An intriguing question that remains open is whether the LexBFS incremental split decomposition algorithm could also be turned into a linear time algorithm. But to circumvent the use of the union-find data-structure, in that case, we are still missing a generalization of good vertex lemma providing a consecutive property.

This has implications for other problems, where linear time was achieved only under the assumption that a representation of the input is provided. This includes the isomorphism problem for circle graphs [23] as well as the partial representation extension problem [4], where one is given a chord diagram for an induced subgraph of the input graph and seeks to extend it to a representation of the full graph. In addition, it may have pave the way for a new linear-time recognition algorithm of circular-arc graphs that follows Hsu's approach [22].

Organization of the paper. Section 2 introduces the basic definitions and presents the model of *graph labelled trees* (GLT) proposed by [18, 20] to represent the split decomposition of a graph. Section 3 is dedicated to circle graphs, their chord diagrams, the data-structures we use to store a circle graph. The GLT representing the split decomposition tree is implemented by a PC-tree [21, 13], and the chord diagrams representing the prime nodes are encoded by Consistent Symmetric Cycles [19]. Section 4 describes LexBFS and corresponding properties of circle graphs, including an alternative proof of the *good vertex lemma*. The original proof of Gioan et al. [19] was two-steps: it first proved the property for prime circle graphs and then the general case follows as a consequence of the recognition algorithm. Our proof is independent of the recognition algorithm. Finally, in Section 5, we present how to adapt the circle graph recognition of Gioan et al. [19] to the new data-structure. The algorithm checks if a vertex ending a LexBFS can be inserted in the split PC-tree of a circle graph and if so updates the split PC-tree representation. Along with the correctness proof, we provide an amortized time complexity analysis based on the method of potentials [32] (see also [5, Ch. 17.3]). Interestingly, we observe that this complexity analysis also applies to the original algorithm [19], and to the split decomposition algorithm [20] as well, and simplifies the complexity therein.

2 Preliminaries and split decomposition

2.1 Basic definitions

A *word* over an alphabet Σ is a finite sequence of letters of Σ. If A is a word over Σ, then A^r denotes its *reversal* (the ordering between the letters of A has been reversed). The concatenation between two words A and B over Σ is a word over Σ denoted AB. A subsequence F of consecutive letters in a word A is called a *factor* of A.

A *circular word* C over Σ is a circular sequence of letters of Σ. It can be represented by a word by considering that the first letter follows the last letter. Observe that such a representation fixes an arbitrary first letter. So if a circular word C is represented by the word AB, then BA also represents C. To denote that equivalence, we write $C \sim AB \sim BA$. Observe that, if C a circular word represented by the word A, that is $C \sim A$, then the reversal of C is $C^r \sim A^r$. The notion of factor naturally extends to circular words: F is a factor of C if there exists a word A such that $C \sim A$ and F is a factor of A.

Unless specified, we will assume that every graph $G = (V, E)$, on vertex set V and edge set E, is connected. The neighborhood of a vertex x in G will be denoted $N_G(x)$ or simply $N(x)$ if the graph is clear from the context. A *clique* is a graph where every vertex is adjacent to every other vertex. A *star* is a tree with one universal vertex. For a graph $G = (V, E)$ and a vertex x not belonging to V, we let denote $G' = G + x$ [2] the graph obtained by adding x to V and all the edges between x and $N_{G'}(x)$ to E. Given a subset S of vertices of V, we let $G[S]$ denote the subgraph induced by S. A *vertex ordering* σ is a total order on the vertices of a graph G. If x is smaller than y in σ, we write $x <_\sigma y$. For a subset S of vertices, we let $\sigma[S]$ denote the subordering of σ induced by the vertices of S, that is for every $x, y \in S$, $x <_{\sigma[S]} y$ if and only if $x <_\sigma y$.

2.2 Split decomposition and graph labelled trees

A bipartition (A, B) of a set V is *non-trivial* if $|A| > 1$, $|B| > 1$.

▶ **Definition 1** (Split of a graph). *A* split *of a graph $G = (V, E)$ is a non-trivial bipartition (A, B) of its vertex set V with two subsets $A' \subseteq A$, $B' \subseteq B$, called* frontiers *of the split, such that for every $x \in A$ and $y \in B$, it holds that $xy \in E$ if and only if $x \in A'$ and $y \in B'$.*

Splitting a graph G with respect to a split (A, B) of G returns two graphs G_A and G_B obtained from G by contracting B into a vertex x_A and A into a vertex x_B (the neighborhood of x_A in G_A is A' and of x_B in G_B is B'), respectively. Let $G = (V, E)$ and $G' = (V', E')$ be two graphs and let $x \in V$ and $x' \in V'$ be two vertices. The *join* between G and G' with respect to x and x', denoted $(G, x) \otimes (G', x')$, is the graph on vertex set $(V \cup V') \setminus \{x, x'\}$ such that two vertices y and z are adjacent if and only if $yz \in E$, or $yz \in E'$, or $y \in N_G(x)$ and $z \in N_{G'}(x')$. So the join and the splitting are reverse operations for large enough graphs. Observe that if G is the single vertex graph, then $(G, x) \otimes (G', x')$ is isomorphic to $G[V \setminus \{x'\}]$, and if G is the clique of size 2, then $(G, x) \otimes (G', x')$ is isomorphic to G' Otherwise, if G and G' are connected, then $(V \setminus \{x\}, V' \setminus \{x'\})$ is a split of $(G, x) \otimes (G', x')$. A graph G is *degenerate* if every non-trivial bipartition of V is a split of G. It is known that if G is degenerate, then G is either a *clique* or a *star*. A graph G is *prime* if it does not contain any split.

[2] Though the neighborhood of x is not specified in the notation $G + x$, it will always be clear from the context.

▶ **Definition 2** (Graph labelled tree). *A* graph labelled tree *(GLT) is a pair* $(T, \mathcal{F})$*, where* T *is a tree and* $\mathcal{F}$ *a set of graphs, called* label graphs*, such that each node* u *of* T *is labelled by the graph* $G(u) \in \mathcal{F}$ *and is further equipped with a bijection* ρ_u *that bijectively maps the edges incident to* u *to the vertices of* $G(u)$*.*

Let $(T, \mathcal{F})$ be a GLT and let u be a node of $(T, \mathcal{F})$. We say that the node u is *prime* if $G(u)$ is a prime graph and that u is *degenerate* if $G(u)$ is degenerate. We let $V(u)$ denote the vertices of $G(u)$, hereafter called *marker vertices*, and $E(u)$ denote the edges of $G(u)$, hereafter called *label edges*. Since a leaf has only one incident edge, we may abusively consider leaves as marker vertices (namely the leaf u is identified with the marker $\rho_u(e)$, of the unique edge e incident to u). If $e = uv$ is a tree-edge of T, then the marker vertices $\rho_u(e)$ and $\rho_v(e)$ are *opposite* marker vertices called the *extremities* of e. Observe that every marker vertex is the extremity of a tree-edge of T.

A graph labelled tree is designed to represent a graph in a compact manner. To explain how, we need to introduce the notion of accessibility, which can be seen as an extension of the adjacency. Two marker vertices q and q' of distinct nodes are *accessible* from one another if there exists a sequence of marker vertices $\langle q = q_1, \dots, q_{2k} = q' \rangle$ of even length such that q_i, q_{i+1} are the extremities of a tree-edge if i is odd and are adjacent marker vertices in some label graph if i is even.

Let $e = uu'$ be a tree-edge of the GLT $(T, \mathcal{F})$. Consider the marker vertex $q \in V(u)$ that is the extremity e. Then, we let $L(q)$ denote the set of leaves of the subtree of $T - e$ not containing u. Among $L(q)$, we distinguish the subset $A(q)$ of leaves accessible from q.

▶ **Definition 3** (Accessiblity graph). *The* accessibility graph $\mathsf{Gr}(T, \mathcal{F})$ *of a GLT* $(T, \mathcal{F})$ *is the graph whose vertices are the leaves of* T *and such that two vertices are adjacent if and only if the corresponding leaves are accessible from one another.*

Let $q' \in V(u')$ be the extremity of e distinct from q. We observe that if e is not incident to a leaf of T, then the bipartition $(L(q), L(q'))$ is a split of $\mathsf{Gr}(T, \mathcal{F})$ and the frontiers of that split are $A(q)$ and $A(q')$. The *node-join* of two non-leaf nodes u and u' in $(T, \mathcal{F})$ returns the GLT $(T', \mathcal{F}')$ obtained by contracting the tree-edge $e = uu'$, labelling the resulting node v by the graph $G(v) = (G(u), q) \otimes (G(u'), q')$, and letting every marker vertex $q \in V(v)$ be the extremity of the same tree-edge as in $(T, \mathcal{F})$. The *node-split* operation of a GLT is the reverse of the node-join operation. Suppose that (A, B) is a split of the graph $G(u)$ for some node u of $(T, \mathcal{F})$. The node-split of u with respect to (A, B) returns the GLT $(T', \mathcal{F}')$ where node u has been replaced by two adjacent nodes u_A and u_B labelled by $G(u_A) = G(u)[A \cup \{q_A\}]$ for some vertex q_A of the frontier B' and $G(u_B) = G(u)[B \cup \{q_B\}]$ for some vertex q_B of the frontier A', respectively. The marker vertices $q_A \in V(u_A)$ and $q_B \in V(u_B)$ are made the extremities of the tree-edge $u_A u_B$. Every other marker vertex is an extremity of the same tree-edge as in $(T, \mathcal{F})$.

▶ **Theorem 4** ([8, 20]). *For any connected graph* G*, there exists a unique GLT, denoted* $\mathbf{ST}(G)$ *and called the* split-tree *of* G*, whose labels are either prime or degenerate, having a minimum number of nodes and such that* $\mathsf{Gr}(\mathbf{ST}(G)) = G$*.*

3 Circle graphs and split PC-trees

3.1 Chord diagrams and circle graphs

A *chord* on a circle is defined by a pair $c = \{c_1, c_2\}$ of distinct points of the circle, called *endpoints*. Let χ be a set of chords no pair of which share a common endpoint. An *expanded chord diagram* on χ is naturally represented by a circular word $\dot{D}$ over the alphabet

$\mathcal{V} = \bigcup_{c \in \chi} \{c_1, c_2\}$. A *chord diagram* D is obtained from the expanded chord diagram $\dot{D}$ by replacing every endpoint $c_i \in \mathcal{V}$ $(i \in \{1, 2\})$ by the corresponding chord $c \in \chi$. Observe that a chord diagram D is a *double occurrence circular word* over the alphabet χ, that is every chord of χ appears exactly twice in D. We may abusively say that each occurrence of a chord c in D is an endpoint of c. A subset S of chords of χ is *consecutive* in D if D contains a factor F such that for every chord $c \in S$, $|c \cap F| = 1$ and for every chord $c' \notin S$, $c' \cap F = \emptyset$. The first and the last chord of the factor F are called the *bookends* of F (or of S). Let a and b be two endpoints of the chord diagram D. Then $D(a, b)$ is the factor of D containing the set of endpoints comprised between a and b (not containing a and b), while $D(b, a)$ contains those comprised between b and a. In other words, we have that $D \sim aD(a, b)bD(b, a)$. Observe that $D(a, b)^r = D^r(b, a)$ and that $D(b, a)^r = D^r(a, b)$.

Let D and D' be chord diagrams on χ and χ', respectively, and let $x \in \chi$ and $x' \in \chi'$. We define the *circle-join* operation between D and D' with respect to $x = \{x_1, x_2\}$ and $x' = \{x'_1, x'_2\}$ as follows:

$$(D, x) \odot (D', x') \sim D(x_1, x_2)D'(x'_1, x'_2)D(x_2, x_1)D'(x'_2, x'_1)$$

The result is a chord diagram on the set $(\chi \cup \chi') \setminus \{x, x'\}$ (see Figure 3). We observe that the circle-join is not a commutative operation: $(D, x) \odot (D', x') \neq (D', x') \odot (D, x)$.

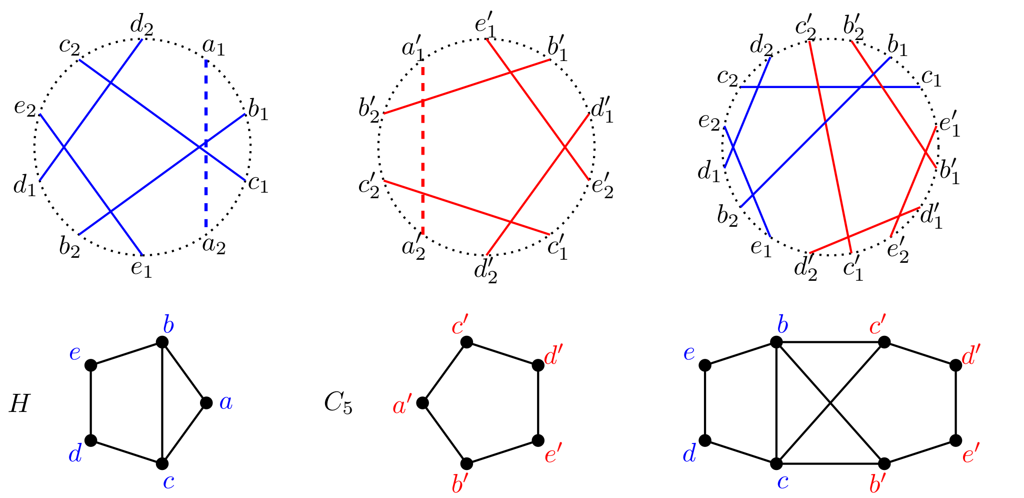

Figure 3 From left to right: the chord diagram D_1 of the house graph H, the chord diagram D_2 of the C_5 and $D_1(a) \odot D_2(a') = D_1(a_1, a_2)D_2(a'_1, a'_2)D_1(a_2, a_1)D_2(a'_2, a'_1)$, the chord diagram of the graph $G = (H, a) \otimes (C_5, a')$ (depicted on the right). Dotted chords represent the vertices (respectively a and a') on which the join is performed. In G, vertices b and c (the neighbors of a in the House graph) are both adjacent to vertices b' and c' (the neighbors of a' in the C_5).

▶ **Lemma 5** ([19, Lemma 3.3]). *Let D and D' be chord diagrams on the sets V and V' of chords, respectively. Let $S \subset V$ and $S' \subset V'$ be consecutive sets of chords in their respective chord diagrams such that $1 < |S| < |V|$ and $1 < |S'| < |V'|$. If x and x' are bookends of S and S', respectively, then $X = (S \setminus \{x\}) \cup (S' \setminus \{x'\})$ is consecutive in (at least) one of the following chord diagrams:*

$$(D, x) \odot (D', x'), \qquad (D', x') \odot (D, x), \qquad (D, x) \odot (D'^r, x'), \qquad (D'^r, x') \odot (D, x).$$

Moreover, the bookends of X are those of S and S' distinct from x and x'.

A *circle graph* $G = (V, E)$ is the intersection graph of a chord diagram D on a set χ of chords of a circle. In other words, there exists a bijection $\rho : V \to \chi$ such that two vertices $x, y \in V$ are adjacent in G if and only if the chords $\rho(x)$ and $\rho(y)$ intersect in D, that is if and only $D(x_1, x_2)$ contains one endpoint among y_1 and y_2 and $D(x_2, x_1)$ the other. We say that D *represents* or *encodes* G. We observe that, in general, a circle graph is encoded by many different chord diagrams. For example, cliques and stars are circle graphs that are represented by many chord diagrams: every clique G on vertex set V is represented by $D \sim AA^r$ where A is an arbitrary permutation of V; every star G with center vertex x is represented by $D \sim xAxA$ where A is an arbitrary permutation of $V \setminus \{x\}$.

▶ **Observation 6.** *Let G and G' be two circle graphs represented by chord diagrams D and D', respectively. Then, for any vertex x of G and any vertex x' of G', the chord diagrams $(D, x) \odot (D', x')$ and $(D', x') \odot (D, x)$ represent the circle graph $H = (G, x) \otimes (G', x')$.*

However, we have the following important property, announced in [15], partially proved in [2], formally proved in [7] (an alternative proof was given in [19]):

▶ **Theorem 7** ([15, 2, 7, 19])**.** *A circle graph $G = (V, E)$ is a prime graph if and only if it has a unique (up to reversal) chord diagram.*

Since degenerate graphs (cliques and stars) are circle graphs, a consequence of Observation 6 is the following well-known characterization of circle graphs.

▶ **Theorem 8** ([15, 2, 26])**.** *A graph G is a circle graph if and only if every prime node u of $\mathbf{ST}(G)$ is labelled by a circle graph $G(u)$.*

3.2 Consistent symmetric cycles

We describe the *consistent symmetric cycle* data-structure (CSC) introduced in [19] that represents chord diagrams and allows to efficiently perform the following operations: consecutive test (see Lemma 9), consecutive preserving join (see Lemma 10) and vertex insertion (see Lemma 11). Let D be a chord diagram on the set χ of chords. The *consistent symmetric cycle* (CSC), denoted $\mathbf{C}(D)$, representing D is a directed graph whose vertices are the chord endpoints of χ, that is $\mu = \bigcup_{x \in \chi} \{x_1, x_2\}$. Every pair of consecutive endpoints in D is linked by two symmetric arcs, hence forming a symmetric directed cycle. It follows that every endpoint has two out-neighbors, which we denote $+_{\mathbf{C}(D)}(\cdot)$ and $-_{\mathbf{C}(D)}(\cdot)$. For every chord x we have the property $+_{\mathbf{C}(D)}(x_1)$ and $+_{\mathbf{C}(D)}(x_2)$ belong to one of the two connected components of $\mathbf{C}(D) \setminus \{x_1, x_2\}$ and $-_{\mathbf{C}(D)}(x_1)$ and $-_{\mathbf{C}(D)}(x_2)$ to the other. To complete $\mathbf{C}(D)$, the two endpoints of every chord are linked with symmetric arcs (see Figure 4).

Let $\mathbf{C}(D)$ be a CSC on the set χ of chords. Let x_1 and x_2 be the two endpoints of a chord x. We let $+_{\mathbf{C}(D)}(x_1, x_2)$ denote the factor of $\mathbf{C}(D)$ obtained by searching $\mathbf{C}(D)$ from $+_{\mathbf{C}(D)}(x_1)$ to $+_{\mathbf{C}(D)}(x_2)$. The factor $-_{\mathbf{C}(D)}(x_1, x_2)$ is defined similarly. In the example of Figure 4, we have $+_{\mathbf{C}(D)}(e_1, e_2) = c_2 d_1 a_1 b_1 c_1 a_2$ and $-_{\mathbf{C}(D)}(e_1, e_2) = d_2 b_2$. We observe that $+_{\mathbf{C}(D)}(x_1, x_2) = +_{\mathbf{C}(D)}(x_2, x_1)^r$ and that either $+_{\mathbf{C}(D)}(x_1, x_2) = D(x_1, x_2)$ or $+_{\mathbf{C}(D)}(x_1, x_2) = D^r(x_1, x_2)$.

The following three lemmas state the complexity of basic operations on CSC's that the algorithm has to perform. Their proofs are straightforward from the data-structure description of CSC's. They have been formally discussed in [19] as Lemmas 5.5–5.7.

▶ **Lemma 9** (Consecutivity test)**.** *Let D be a chord diagram of a circle graph $G = (V, E)$. Given a subset S of vertices of G and $\mathbf{C}(D)$ the CSC representing D, we can test in $O(|S|)$-time if S is consecutive in D.*

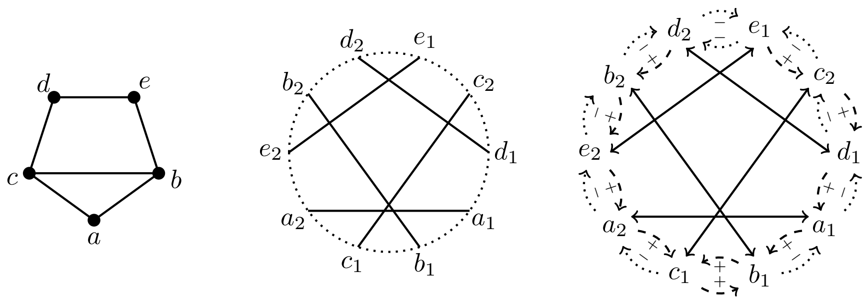

Figure 4 The house graph on the left. On the right, the CSC $\mathbf{C}(D)$ of the (unique) chord diagram D of the house graph (drawn in the middle). The dashed arc leaving an endpoint x represents $+_{\mathbf{C}(D)}(x)$ and the dotted arc represents $-_{\mathbf{C}(D)}(x)$. For example, $-_{\mathbf{C}(D)}(a) = e$ and $+_{\mathbf{C}(D)}(a) = c$.

▶ **Lemma 10** (Consecutivity preserving join). *Let D and D' be chord diagrams of the circle graphs $G = (V, E)$ and $G' = (V', E')$, respectively, and let S and S' be consecutive chords in D and D', respectively. Let further x and y denote the bookends of S, and x' and y' the bookends of S', and $\mathbf{C}(D)$ and $\mathbf{C}(D')$ the CSCs representing D and D'. Then a CSC representing a chord diagram of $(G, x) \otimes (G', x')$ such that $(S \setminus \{x\}) \cup (S \setminus \{x'\})$ is consecutive and has bookends y and y' can be computed in $O(1)$ time.*

▶ **Lemma 11** (Chord insertion). *Let D be a chord diagram of a circle graph $G = (V, E)$ and let S be a subset of vertices of G that is consecutive in D. Given $\mathbf{C}(D)$ the CSC representing D and the bookends b and b' of the factor of D certifying that S is consecutive, a CSC of $G + x$ where the neighborhood of x is S can be computed in $O(1)$ time.*

3.3 PC-trees

The data structure we use to encode the split-tree $\mathbf{ST}(G)$ of a circle graph G is based on PC-trees [21, 13]. We call it the *split PC-tree* of G and it is denoted $\mathbf{PC}(G)$ (see Figure 5 for an example). It stores all marker vertices of $\mathbf{ST}(G)$ and encodes the structure of $\mathbf{ST}(G)$ on top of them as follows. We select an arbitrary leaf of $\mathbf{ST}(G)$ as the root of $\mathbf{PC}(G)$, which we denote by r. Let u be a node of the split-tree $\mathbf{ST}(G)$. For each marker vertex $q \in V(u)$, we store an outgoing arc to its opposite q', which is a marker vertex of $G(u')$ for some node u' adjacent to u in $\mathbf{ST}(G)$. Observe that this way, each tree-edge e of $\mathbf{ST}(G)$ corresponds to a pair of oppositely oriented arcs. We assume moreover that for an arc a of $\mathbf{PC}(G)$, the opposite arc, denoted $\bar{a}$, can be accessed in $O(1)$-time. The way marker vertices of $V(u)$ are stored depends on the type of the node u in $\mathbf{ST}(G)$.

- Suppose u is degenerate. Then u is represented in $\mathbf{PC}(G)$ by a node object $\mathbf{n}(u)$ that stores (i) the type of u, (ii) a (doubly-linked) list $\mathbf{L}(u)$ of the marker vertices in $V(u)$ (iii) a pointer to the node of $V(u)$ whose arc points towards the root r, and (iv) in case u is a star, a pointer to the marker vertex that is the center of the star. Further, each marker vertex of $V(u)$ is equipped with a pointer to the node object $\mathbf{n}(u)$.
- Suppose u is not degenerate. Then $\mathbf{PC}(G)$ does not store any node object for u. Instead, the vertices of $V(u)$ are stored in a CSC $\mathbf{C}(u)$ representing a chord diagram of $G(u)$. For each vertex $x \in V(u)$, one of the two endpoints of the corresponding chord is made incident to the symmetric pointers representing the tree-edge e of which x is the extremity. Further, a flag is used to mark the vertex whose arc points towards the root r of $\mathbf{ST}(G)$.

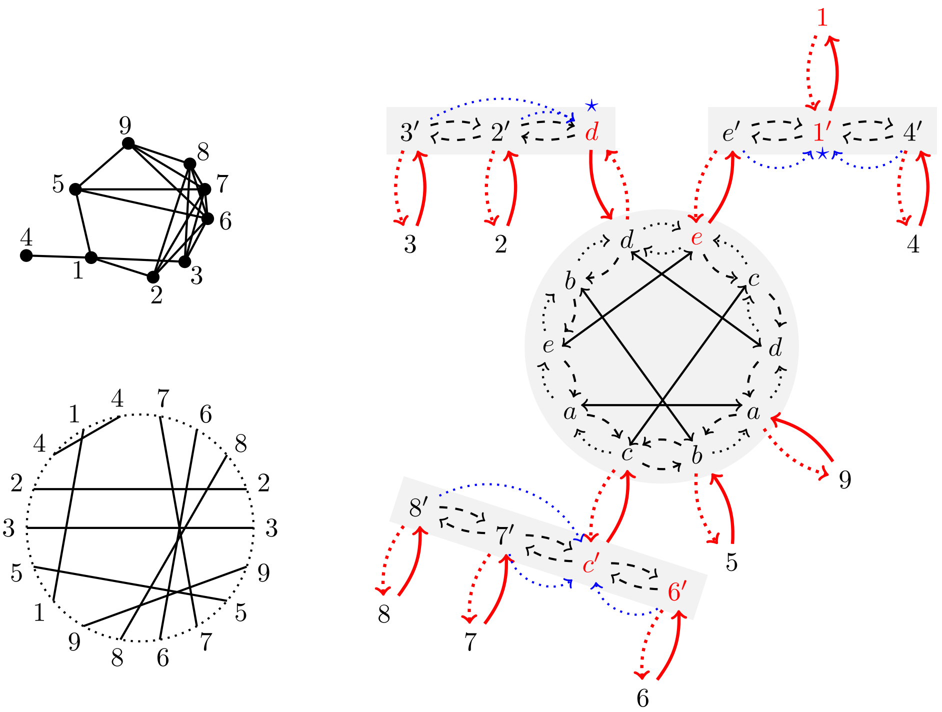

Figure 5 A circle graph G, a chord diagram of G and the split PC-tree $\mathbf{PC}(G)$ rooted at leaf 1. The red plain arcs correspond to the tree-edges from nodes to their parent. The split-tree $\mathbf{ST}(G)$ contains a unique prime node u labelled by the house graph and represented by a CSC. The *root of a node* u is the extremity of the arcs between u and its parent. Roots are marker vertices or the endpoints colored red. Every marker vertex of a degenerate node has a pointer towards its root. In a star node u, the center is identified by a blue star (which can be distinct from the root of u).

Observe that, given any marker vertex from $V(u)$ of some degenerate node u, we can determine the corresponding node object and thus also its parent arc in $O(1)$ time. On the other hand, due to the lack of a node object to represent a non-degenerate node u, finding the parent arc of u is a costly operation. Indeed, given a marker vertex from $V(u)$, one has to traverse $\mathbf{C}(u)$ until the endpoint is found whose flag indicates that it is incident to the parent arc. On the positive side, by Lemma 10, given two CSC's, it is possible to perform a node-join operation in $O(1)$ time and since the result is always a non-degenerate node, there is no need to update pointers to a node object, especially the pointer to the parent of the resulting node.

4 Lexicographic Breadth-First-Search

The algorithm LexBFS (Lexicographic Breadth-First-Search) was introduced by Rose, Tarjan and Lueker [30] to recognize chordal graphs in linear time. It has since been used in many different contexts. We refer to [20] and references therein for a description of LexBFS. The circle graph recognition algorithm uses LexBFS as a preprocessing step to compute an ordering of the vertices of the input graph G. That ordering is then used to incrementally build $\mathbf{PC}(G)$ if G is a circle graph by adding one vertex at a time.

A *LexBFS ordering* of a graph G is an ordering σ in which LexBFS has visited the vertices of G. Let y be a vertex of G. The *slice* $S(y)$ is the largest factor of σ starting at y such that for every $x \prec_\sigma y$, x is adjacent to y if and only if x is adjacent to every vertex of $S(y)$. It is known that the restriction $\sigma[S(y)]$ to the vertices of the slice $S(y)$ is a LexBFS ordering of $G[S(y)]$ [6]. A vertex x of a graph G is *good* if there exists a LexBFS ordering σ whose last vertex is x. Due to the following lemma, good vertices are crucial to the LexBFS incremental recognition algorithm of circle graphs. This lemma is stated and proved in [19] for the case of prime graphs. The case of arbitrary graph follows from the algorithm of [19]. We provide a self-contained proof independent from their algorithm.

▶ **Lemma 12** (Good vertex lemma). *Let G be a circle graph. If x is a good vertex of G, then G has a chord diagram D in which $N(x)$ is consecutive.*

Proof. Let σ be a LexBFS ordering of G ending at x. Let z and z' be the first two vertices of σ in order. We prove the statement by induction on the number of slices containing x. Without loss of generality, we may assume that G is connected.

Suppose that for every vertex $w \notin \{z, z', x\}$, $S(w)$ is not a slice containing x. For the sake of contradiction, assume that $N(x)$ is not consecutive in any chord diagram of $G = (V, E)$. Observe that in every chord diagram D, either one endpoint of z appears in $D(x_1, x_2)$ and the other in $D(x_2, x_1)$, or the two endpoints of z appear in one of $D(x_1, x_2)$ and $D(x_2, x_1)$. Without loss of generality, suppose that $D(x_1, x_2)$ contains at most one endpoint of z. Since $N(x)$ is not consecutive, at least one chord has its two endpoints in $D(x_1, x_2)$. Amongst such chords, let y be the one occurring the earliest in σ. Observe that by the choice of y, every neighbor v of y such that $v <_\sigma y$ is adjacent to x. Since $y <_\sigma x$, we also have that every non-neighbor of v' of y is a non-neighbor of x and thereby $x \in S(y)$. Observe moreover that, by assumption on $D(x_1, x_2)$, we have $y \neq z$. It follows that if $y \neq z'$, we have a contradiction. So assume that $y = z'$. Observe that by connectivity of G, we have that z is universal in G and thereby its chord has one endpoint in $D(x_1, x_2)$ and the other in $D(x_2, x_1)$. Then again, since $N(x)$ is not consecutive, at least one chord has its two endpoints in $D(x_2, x_1)$. Amongst such chords, let y' be the one occurring the earliest in σ. By the same argument than before we can prove that $S(y')$ is a slice containing x and moreover $y' \notin \{z, z', x\}$, which is a contradiction.

Let us assume the property holds if the last vertex of a LexBFS ordering belongs to $k \geq 2$ slices. Suppose that x belongs to $k+1$ slices of σ with $k \geq 2$. Let y be the vertex of G occurring last in σ such that $y \notin \{z, z', x\}$ and $x \in S(y)$. If such a vertex does not exist, then by the discussion above we are done. Let us consider $B = \{v \in V \mid v \leq_\sigma y\}$ and $A = S(y) \cup \{y'\}$ where y' is a neighbor of y such that $y' <_\sigma y$. Observe that $\sigma[B]$ and $\sigma[A]$ are LexBFS orderings of $G[B]$ and $G[A]$ respectively ending at y and x. Since $\sigma[B]$ has fewer slices containing y than σ has containing x, by the induction hypothesis, $G[B]$ has a chord diagram D_B in which $N(y) \cap B$ is consecutive. Also the only slices containing x in $\sigma[A]$ are $S(v) = A$ and $S(x) = \{x\}$. Again by induction hypothesis, $G[A]$ has a chord diagram D_A in which $N(x) \cap A$ is consecutive. Hence, since y' is universal in G_A, we may assume that $D_A(x_1, x_2) = X \cdot y'_2 \cdot X'$ certifies the consecutiveness of $N(x) \cap A$. Observe then that $D_A(y'_1, y'_2) \odot D_B(y_1, y_2) = D_A(y'_1, y'_2) \cdot D_B(y_1, y_2) \cdot D_A(y'_2, y'_1) \cdot D_B(y_2, y_1)$ is a chord diagram of G in which $N(x) = (N(x) \cap A) \cup (N(y \cap B)$ is consecutive. ◀

Let σ be a LexBFS ordering of a graph G and let u be a node of a GLT $(T, \mathcal{F})$ such that $\mathsf{Gr}(T, \mathcal{F}) = G$. Then we can define from σ a LexBFS ordering σ_u of $G(u)$ in the following way [20]: for two marker vertices q and q' of $G(u)$, let us denote x the earliest vertex of $A(q)$ in σ and x' the earliest vertex of $A(q')$ in σ, then $q <_{\sigma_u} q'$ if and only if $x <_\sigma x'$.

▶ **Lemma 13** ([20]). *Let σ be a LexBFS ordering of a graph G and let u be a node of $\mathbf{ST}(G)$. Then σ_u is a LexBFS ordering of $G(u)$.*

To illustrate Lemma 13, consider the graph G of Figure 5. The numbering of its vertices from 1 to 9 is a LexBFS ordering. Considering the unique prime node u of $\mathbf{ST}(G)$, then $\sigma_u = \langle e, d, b, c, a \rangle$ is a LexBFS ordering of the house graph $G(u)$.

Lemma 12, Lemma 13 together with Lemma 10 allows us to sketch a circle graph recognition algorithm. The idea is to iteratively process the vertices of an input graph G according to a LexBFS ordering in order to build $\mathbf{PC}(G)$. By Lemma 12, at each step, if G is a circle graph, the neighborhood of the vertex x to be inserted is consecutive in some chord diagram of the so-far processed subgraph. By Lemma 13, this property is valid for the label graph of every node the current split PC-tree. Since adding a vertex may kill some existing split, we may have to perform node-join operations to compute the updated split PC-tree. By Lemma 10, this can be done while preserving the consecutiveness of the neighborhood of x. The challenge is to identify the part of the current split PC-tree that has to be shrunk into a single node. As we will see, either the input graph G is a circle graph and this can be done efficiently, or we are able to conclude that G is not a circle graph.

5 The vertex insertion algorithm

Throughout this section, we assume that $G = (V, E)$ is a circle graph, that $\mathbf{PC}(G)$ is the split PC-tree of G encoding its split-tree $\mathbf{ST}(G) = (T, \mathcal{F})^3$. We consider a new vertex x with neighborhood $S \subseteq V$ that is the last vertex of a LexBFS ordering of $G' = G + x$, that is, x is good in the graph G'. We assume without loss of generality that G is connected and $S \neq \emptyset$.

We let $T(S)$ denote the minimal subtree of T that contains all vertices in S. Let q be a leaf of T or a marker vertex (possibly an endpoint) of some node u of T. Following the work of Gioan et al. [20, 19], the state of q (with respect to S) is *perfect* if $S \cap L(q) = A(q)$; *empty* if $S \cap L(q) = \emptyset$; and *mixed* otherwise (see Figure 6). We let $P(u)$ denote the set of perfect marker vertices of u and $MP(u)$ the set of perfect or mixed marker vertices of u. For a degenerate node u, we define:

$$P^*(u) = \{q \in V(u) \mid q \text{ is perfect and not the center of a star}\},$$

$$E^*(u) = \{q \in V(u) \mid q \text{ is empty, or } q \text{ is perfect and the center of a star}\}.$$

A node u of T is *hybrid* if every marker vertex q of u has the property that q is perfect or empty and its opposite is mixed. A tree-edge is *mixed* if both its extremities are mixed. It can be proved [20] that the subtree T' of T defined by the mixed tree-edges is unique and connected, it is called the *fully-mixed subtree* of $\mathbf{ST}(G)$. The following lemma is central to the correctness of the circle graph recognition algorithm:

▶ **Lemma 14** ([19]). *Let $G' = G + x$ be a prime graph such that x is a good vertex of G' and G is a circle graph. Then G' is a circle graph if and only if for every node u in $\mathbf{ST}(G)$, marked with respect to $N_{G'}(x)$, $G(u)$ has a chord diagram in which $MP(u)$ is consecutive, with the mixed marker vertices being bookends.*

Let us discuss the above statement and provide some intuition of its correctness. First if G' is a circle graph, then by Theorem 7, it has a chord diagram D', which is unique up to reversal. Since x is good, by Lemma 12, $N_{G'}(x)$ is consecutive in D'. Observe that $N_{G'}(x)$ is still consecutive in the chord diagram D of G obtained by removing the chord x from D'.

[3] Given that we can transform $\mathbf{ST}(G)$ into $\mathbf{PC}(G)$ and vice versa, depending on the context, if more convenient we may abusively switch between these two objects.

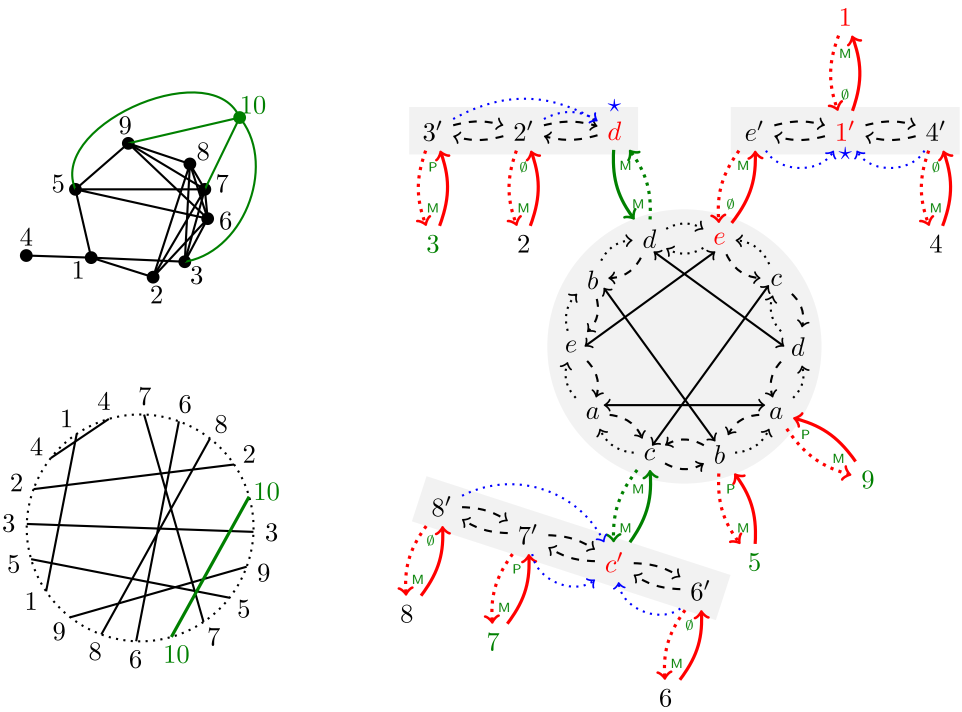

Figure 6 A circle graph $G' = G + 10$ with vertex 10 being a good vertex of G' adjacent to $S = \{3, 5, 7, 9\}$, a chord diagram of G' and the split PC-tree $\mathbf{PC}(G)$ marked with respect to $\{3, 5, 7, 9\}$. Every marker vertex is assigned green tags: P if it is perfect; M if it is mixed; $\emptyset$ otherwise. The two green tree-edges are mixed and $\mathbf{PC}(G)$ does not contain any hybrid node. Thereby endpoint e is marked empty since $L(e) = \{1, 4\}$ does not intersect S; the endpoint a is marked perfect since $L(a) = \{9\} = A(a) \subseteq S$; and the endpoint c is marker mixed since $L(c) = \{6, 7, 8\} = A(c)$ while $S \cap L(c) \neq A(c)$.

However, G may not be a prime graph. If so we may retrieve the chord diagrams of every prime node of $\mathbf{ST}(G)$ by performing a series of *circle-split* operations, the reverse operation of circle-join. When doing so, we can observe that the consecutive property is preserved as indicated in Lemma 14. Conversely, assume that every node of $\mathbf{ST}(G)$ has a chord diagram satisfying the property stated in Lemma 14. Then by Lemma 5, iteratively performing for every tree-edge uv, a circle-join between the chord diagrams $D(u)$ and $D(v)$ with respect to the extremities of the tree-edge uv eventually returns a chord diagram D of G in which $N_{G'}(x)$ is consecutive. It is then possible to insert in D a chord x crossing exactly $N_{G'}(x)$, certifying that $G' = G + x$ is a circle graph.

As announced earlier, the recognition algorithm iteratively inserts vertices according to a LexBFS ordering. Of course, for the sake of time complexity, we cannot afford at each insertion step contracting $\mathbf{ST}(G)$ in a single node. Lemma 14 allows us to maintain a chord diagram for each prime node providing local certificates of membership to the class of circle graphs.

5.1 Updating the split-tree

Gioan *et al.* [20, Theorem 4.14, Proposition 4.15–4.20] show that exactly one of the following cases occurs and in each case describe how to obtain the split-tree $\mathbf{ST}(G')$, with $G' = G + x$, from $\mathbf{ST}(G) = (T, \mathcal{F})$.

1. *T contains a clique node u whose marker vertices are all perfect; or T contains a star node u whose marker vertices are all empty except its center, which is perfect; or T contains a unique hybrid node u, which is prime.*

   In each of these cases, the node u is unique and the split-tree $\mathbf{ST}(G')$ is obtained by adding to T a leaf ℓ_x adjacent to u, adding to $G(u)$ a new marker vertex q_x adjacent to $P(u)$ and opposite to ℓ_x.
   We observe that the type of the updated node u is the same as in $\mathbf{ST}(G)$. This implies that in the case u is degenerate, by Theorem 8, $G+x$ is a circle graph. So let us discuss the case that u is a prime node. It is then represented in $\mathbf{PC}(G)$ by a CSC $\mathbf{C}(u)$ encoding its chord diagram, which is unique. In $\mathbf{C}(u)$, for $G+x$ to be a circle graph, $MP(u)$ has to be consecutive with the mixed marker vertices being the bookends (see Lemma 14). If so, adding q_x consists in inserting a new chord whose extremities become the new bookends of $P(u)$. Otherwise, we can conclude that $G+x$ is not a circle graph.

2. *T contains a tree-edge e whose extremities are both perfect, and this edge is unique; or T contains a tree-edge with one perfect and one empty extremity, and this edge is unique.*

   In this case the tree-edge e is unique and $\mathbf{ST}(G')$ is obtained by: i) subdividing e with a new node u_x; ii) adding a leaf ℓ_x adjacent to u_x; and iii) labelling u_x by a clique of size 3 (if both extremities of e are perfect) or a star on two leaves whose center is opposite to the extremity of e that is empty. Observe that since the new node u_x is degenerate, by Theorem 8, $G+x$ is a circle graph.

3. *T contains a hybrid node u, and this node is degenerate.*

   In this case the node u is unique and $\mathbf{ST}(G')$ is obtained as follows. First perform a node-split on u according to the split $(P^*(u), E^*(u))$, creating a new tree-edge e, whose extremities are both perfect or one is perfect and the other is empty. Then e can be processed as in the previous case. Again, since the modified nodes and the new node are degenerate, by Theorem 8, $G+x$ is a circle graph.

4. *T contains a (unique) fully mixed subtree M.*

   In this case, the fully mixed subtree is first cleaned and then contracted into a single node v by performing node-joins. When finally adding x, the graph $G(v)$ becomes a prime graph. Along the series of the node-joins, we need to compute chord diagrams that preserve the consecutive property of $N_{G'}(x)$ (see Lemma 10). More precisely, the split-tree $\mathbf{ST}(G')$ is obtained in three steps as follows:

   (i) First, we clean the fully mixed subtree by performing on each degenerate node u of M a node-split with respect to the splits $(P^*(u), V(u) \setminus P^*(u))$ and/or $(E^*(u), V(u) \setminus E^*(u))$.

   The resulting GLT is denoted $\mathsf{cl}(\mathbf{ST}(G))$. Observe that the set of mixed tree-edges is left unchanged and thereby M can still be abusively considered as the fully mixed subtree of $\mathsf{cl}(\mathbf{ST}(G))$. The difference with $\mathbf{ST}(G)$ is now that every degenerate node of M contains at most one perfect and at most one empty marker (see Figure 7). Moreover by Lemma 14, if $G+x$ is a circle graph, every node u of M has at most two mixed marker vertices which form the bookends of the factor certifying the consecutiveness of $MP(u)$. It follows that the degenerate nodes of $\mathsf{cl}(\mathbf{ST}(G))$ have bounded degree and each of them can be equipped with the accurate CSC in constant time. Otherwise, we can conclude that $G+x$ is not a circle graph.

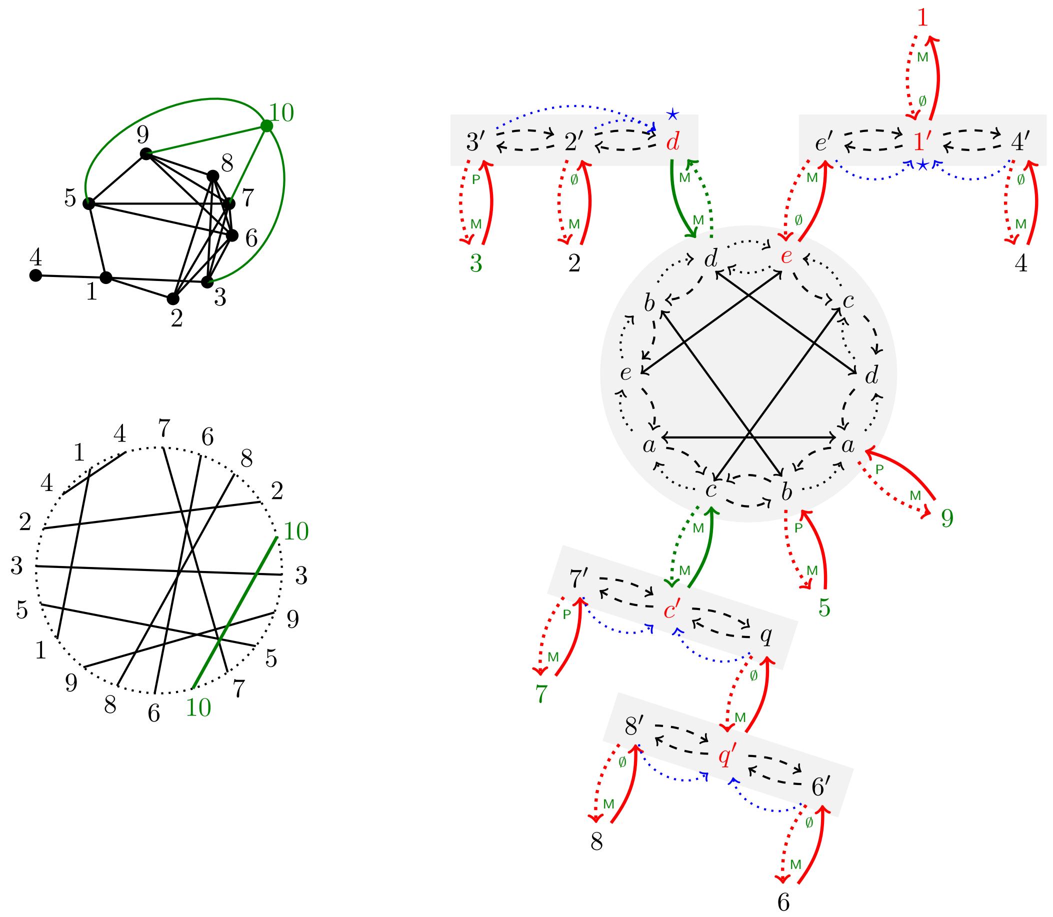

Figure 7 A circle graph $G' = G + 10$ with vertex 10 being a good vertex of G' adjacent to $\{3, 5, 7, 9\}$, a chord diagram of G' and the cleaned split PC-tree $\mathsf{cl}(\mathbf{PC}(G))$ marked with respect to $\{3, 5, 7, 9\}$.

(ii) Then, we contract M into a single node v by performing for every mixed tree-edge $e = uu'$ in M a node-join on u and u'.

As discussed above, each node-join has to return a chord diagram that preserves the consecutive property of $N_{G'}(x)$. Testing if such a chord diagram exists and computing it, can be done efficiently using the CSC data-structure (see Lemma 10). If there is no such chord diagram, then the algorithm stops and concludes that the graph $G + x$ is not a circle graph.

(iii) Finally, in the contracted GLT, we attach a leaf ℓ_x adjacent to node v; and we add to $G(v)$ a new marker q_x adjacent to the marker vertices of $P(u)$ and opposite to ℓ_x.

In this last step, adding q_x consists in inserting a new chord in the chord diagram of $G(v)$, which is possible since it satisfies the consecutive property of $N_{G'}(x)$ (see Lemma 11).

5.2 Recognition algorithm

Given that $\mathbf{ST}(G)$ (or $\mathbf{PC}(G)$) has been correctly marked with respect to $N_{G'}(x)$, where $G' = G + x$, the correctness of the updating algorithm described in Subsection 5.1 is proved by Gioan *et al.* [19] and has been discussed along its description. However, its complexity analysis relies on the split PC-tree data-structure. To achieve an overall linear running time, we implement the updating algorithm in such a way that each vertex x is processed in amortized $O(|N_{G'}(x)|)$ time. Compared to the works of Gioan et al. [20, 19], the main challenge is that, given a marker vertex q that belongs to a prime node u, our split PC-tree data structure does not allow to efficiently determine the parent of u. Using the fact that such nodes are prime, we have a chord diagram for them, and x is inserted according to a LexBFS order allows us to anyway determine the parent without sacrificing the running time at least in all cases where this information is needed. Using the information acquired in the first step, the second step can be implemented in much the same way as in the work of Gioan et al. [19]. The major difference is that, in contrast to the data structure they use, ours allows to implement the contraction of fully mixed edges in Case 4 in constant time as there is no need to propagate the parent pointer information. Let us recall that, given a graph $G' = G + x$ such that x is a good vertex and G is a circle graph, and given the split PC-tree $\mathbf{PC}(G)$ encoding $ST(G) = (T, \mathcal{F})$, the insertion algorithm works in two steps:

1. compute $T(N_{G'}(x))$, the minimal subtree of T covering $N_{G'}(x)$;
2. identify the cases according to the description of Subsection 5.1 and update the split-tree correspondingly.

We start discussing the implementation of the first step. For the sake of complexity efficiency, as stated by Lemma 15, if G' is a circle graph, we guarantee to compute $T(N_{G'}(x))$ in the expected time complexity. Otherwise, the algorithm may be able to conclude, already at that step, that $G' = G + x$ is not a circle graph. The reader has to keep in mind that if $\mathbf{PC}(G)$ encodes the split-tree $\mathbf{ST}(G) = (T, \mathcal{F})$, the tree T is not explicitly stored in $\mathbf{PC}(G)$ since we are lacking node objects to represent the prime nodes of $\mathbf{ST}(G)$. However the tree-edges of T are present in $\mathbf{PC}(G)$. The algorithm identifies these tree-edges.

▶ **Lemma 15.** *Let $G' = G + x$ be a graph such that x is a good vertex of G' and G is a circle graph. Let $\mathbf{PC}(G)$ be the split PC-tree of G encoding the split-tree $\mathbf{ST}(G) = (T, \mathcal{F})$. In time $O(|T(N_{G'}(x))|)$, we can either compute $T(N_{G'}(x))$ or conclude that G' is not a circle graph.*

Proof. We first describe the algorithm and then analyze its complexity. We let $\mathcal{L}(G)$ denote the leaves of $\mathbf{PC}(G)$, $\mathcal{D}(G)$ the set of node objects representing the degenerate nodes of $\mathbf{ST}(G)$ in $\mathbf{PC}(G)$, and $\mathcal{C}(G)$ the set of chord endpoints that are involved in the chord diagrams representing the prime nodes of $\mathbf{ST}(G)$. We set $\mathcal{R}(G) = \mathcal{L}(G) \cup \mathcal{D}(G) \cup \mathcal{C}(G)$. To identify the tree-edges of $T(N_{G'}(x))$, we search $\mathbf{PC}(G)$ through the set $\mathcal{R}(G)$. The algorithm is the following:

1. Initially every leaf in $N_{G'}(x)$ is marked as *active*. Every element of $\mathcal{R}(G) \setminus N_{G'}(x)$ is considered as *inactive*. The set of active elements of $\mathcal{R}(G)$ is managed in a queue $\mathsf{Active}(G)$. In parallel, we maintain a set of *visited* elements of $\mathcal{R}(G)$, denoted $\mathsf{Visited}(G)$. Recall that the root r of $\mathbf{PC}(G)$ is a leaf.
2. We perform a first bottom-up traversal of $\mathbf{PC}(G)$ from the leaves in $\mathsf{Active}(G)$. The objective of this step is to identify a set $\mathfrak{E}$ of tree-edges containing those of $T(N_{G'}(x))$. As long as either $r \notin \mathsf{Visited}(G)$ and $|\mathsf{Active}(G)| \geq 2$, or $r \in \mathsf{Visited}(G)$ and $\mathsf{Active}(G) \neq \emptyset$, we pick the head element $a \in \mathsf{Active}(G)$, remove it from $\mathsf{Active}(G)$ and add it to $\mathsf{Visited}(G)$. If $a \neq r$, there are three cases to consider:
   - $a \in \mathcal{L}(G)$: Let q be the opposite of a. The tree-edge whose extremities are a and q is added to $\mathfrak{E}$. Observe that q is either a marker vertex of a degenerate node or q is the extremity of a chord of the chord diagram representing a prime node). In the former case, if $u \notin \mathsf{Active}(G) \cup \mathsf{Visited}(G)$, then add u to $\mathsf{Active}(G)$. In the case that $q \in \mathbf{C}(G)$, if $q \notin \mathsf{Active}(G) \cup \mathsf{Visited}(G)$, then we add q to $\mathsf{Active}(G)$.

- $a \in \mathcal{D}(G)$: Let p be the root marker vertex of the degenerate node a and q be the opposite of p. We proceed with q as in the previous case.
- $a \in \mathcal{C}(G)$: Let $\{a, a'\}$ be the corresponding chord. Let u denote the node of $\mathbf{ST}(G)$ containing that chord and $\mathbf{C}(u)$ be the corresponding CSC stored in $\mathbf{PC}(G)$. For each chord endpoint b that is consecutive to a or a' in $\mathbf{C}(u)$, we check if b is the flagged root of $\mathbf{C}(u)$. If so, let q be b's opposite. We proceed with q as in the previous cases. Moreover we add to $\mathfrak{E}$ a so-called *fake tree-edge* whose extremities are a and b to later retrieve this flagged root information.

3. We perform a second bottom-up traversal of $\mathbf{PC}(G)$ from the leaves in $N_{G'}(x)$. The objective is to link the identified tree-edges of $\mathfrak{E}$. Observe that during the previous search, when visiting an endpoint of a chord in a CSC $\mathbf{C}(u)$, the flagged root b of $\mathbf{C}(u)$ may or may not have been identified. Suppose it was when visiting the endpoint a. Then a fake tree-edge between a and b was added to $\mathfrak{E}$. We now search from a the set of consecutive endpoints that were visited during the previous search. For each of them, we add to $\mathfrak{E}$ a new fake tree-edge to a.
At this point, we have computed a forest whose edges are $\mathfrak{E}$. Observe that in this implementation, if the root endpoint b of a CSC $\mathbf{C}(u)$ is identified, then node u of $\mathbf{ST}(G)$ is represented by a star whose root is b and possibly a set of isolated endpoints.
4. The last step consists in checking whether $\mathfrak{E}$ allows to retrieve $T(N_{G'}(x))$. We first check whether the resulting set $\mathfrak{E}$ of (fake) tree-edges form a connected tree. If not, we return that G' is not a circle graph. Otherwise, let $T(\mathfrak{E})$ denote that tree. We may have to clear the path attached to the root s of $T(\mathfrak{E})$. Suppose that s is distinct from the root of $\mathbf{PC}(G)$ or it is the root r of $\mathbf{PC}(G)$ (which is a leaf of $\mathbf{PC}(G)$) but the corresponding vertex does not belong to $N_{G'}(x)$. Then remove from $T(\mathfrak{E})$, the tree-edges of the path from that root to its first descendant with at least two children. This step can be completed by an extra search.

Since the described algorithm basically consists in three traversals, its complexity is $O(|\mathfrak{E}|)$. The key point concerning the running time is that the round-robin execution of the upward searches in step 2 guarantees that $|\mathfrak{E}| \leq 2 \cdot |T(N_{G'}(x))|$.

Concerning the correctness, the important point is that, if G' is a circle graph and the marker corresponding to the parent of a prime node u is in $MP(u)$, then by Lemma 14, one of its consecutive chords (in CSC(u)) is also in $MP(u)$, and thus the parent chord will be identified at some point during the traversal. Thus, if G' is a circle graph the algorithm correctly computes the tree-edges of $T(N_{G'}(x))$. ◀

We now have to mark the tree-edge extremities of $\mathbf{PC}(G)$, each of which is either a marker vertex of a degenerate node or an endpoint of a chord in a CSC. Given that $T(N_{G'}(x))$ has been identified, the marking procedure that is fully described in [20] (Lemma 5.10 therein) runs in $O(|T(N_{G'}(x))|)$ time. Within the same time complexity, we can identify which of the cases described in Subsection 5.1 holds. At that step, the node-split operation on degenerate nodes is required. Let us remark that, as in a split PC-tree every degenerate node maintains a pointer to its parent node, we can perform the node-split as described in [20].

▶ **Lemma 16** ([20]). *Let u be a degenerate node of the split PC-tree $\mathbf{PC}(G)$ of a circle graph. If (A, B) is a split of $G(u)$, then node-splitting u according to (A, B) can be done in $O(|A|)$-time.*

Let us now describe how, having computed $T(N_{G'}(x))$ and marked $\mathbf{PC}(G)$ according to $N_{G'}(x)$, we can efficiently compute $\mathbf{PC}(G + x)$.

▶ **Lemma 17.** *Let $G' = G + x$ be a graph such that x is a good vertex of G' and G is a circle graph. Assume that $\mathbf{PC}(G)$ and $T(N_{G'}(x))$ are given and that every edge extremity is marked according to $N_{G'}(x)$. In time $O(|T(N_{G'}(x))|)$, we can either compute $\mathbf{PC}(G')$ or conclude that G' is not a circle graph.*

Proof. Let us consider the four distinct cases identified in Subsection 5.1.

1. *T contains a clique node u whose marker vertices are all perfect; or T contains a star node u whose marker vertices are all empty except its center, which is perfect; or T contains a unique hybrid node u, which is prime.*

   Suppose that u is prime, since otherwise $\mathbf{PC}(G')$ is obtained by adding a leaf to a degenerate node which can be done in $O(1)$-time. We need to test if the chord endpoints of $MP(u)$ are consecutive in the CSC $\mathbf{C}(u)$. This can be done in $O(|MP(u)|)$-time by Lemma 9. If the test fails, we can conclude that $G + x$ is not a circle graph. Otherwise we insert the new chord corresponding to x, which by Lemma 11 can be done in $O(1)$-time. Observe that if a marker vertex (or a chord endpoint) is perfect or mixed, then its incident tree-edge belongs to $T(N_{G'}(x))$. So we have $|MP(u)| \leqslant |T(N_{G'}(x))|$ implying that the global complexity is $O(|T(N_{G'}(x))|)$.

2. *T contains a tree-edge e whose extremities are both perfect, and this edge is unique; or T contains a tree-edge with one perfect and one empty extremity, and this edge is unique.*

   In this case, $\mathbf{PC}(G')$ is obtained by subdividing e by a new ternary degenerate node. This can clearly be achieved in $O(1)$-time.

3. *T contains a hybrid node u, and this node is degenerate.*

   In this case, $\mathbf{PC}(G')$ is obtained by first performing a node-split of u according to $(P^*(u), E^*(u))$ and then inserting a new degenerate node. By Lemma 16 and the discussion above, this can be done in $O(|P(u)|)$-time. Since every perfect marker vertex of u is incident to a tree-edge of $T(N_{G'}(x))$, the statement holds.

4. *T contains a (unique) fully mixed subtree M.*

   We first have to perform the cleaning step. By Lemma 16, this can be done in global $O(|T(N_{G'}(x))|)$ time. Then we construct a CSC for each mixed degenerate node, which takes global $O(|T(N_{G'}(x))|)$ time, since each of them has constant size. Finally, by Lemma 10, the contraction of the fully mixed subtree of $\mathsf{cl}(\mathbf{PC}(G))$ can also be performed in $O(|T(N_{G'}(x))|)$-time. Observe that the contraction step may fail if some node join cannot return a chord diagram satisfying the consecutive property. Finally, by Lemma 11, inserting a new chord to the CSC resulting from the series of contraction can be achieved in $O(1)$-time. ◀

5.3 Amortized time complexity analysis

A key ingredient in the complexity analysis is the following result, which bounds the size of $T(N_G(x))$ in terms of the size of $N_G(x)$.

▶ **Lemma 18** ([30, 33]). *Let $\mathbf{ST}(G) = (T, \mathcal{F})$ be the split-tree of a graph G. For every vertex x of G, $|T(N_G(x))| \leq 2 \cdot |N_G(x)|$.*

Given the split PC-tree $\mathbf{PC}(G)$ of a circle graph G, testing if $G' = G + x$ is a circle graph amounts to computing $\mathbf{PC}(G')$. For this we have access to $T(N_{G'}(x))$, with $\mathbf{ST}(G) = (T, \mathcal{F})$, and not to $T'(N_{G'}(x))$, with $\mathbf{ST}(G') = (T', \mathcal{F}')$. Observe that Lemma 17 establishes an insertion complexity in time linear in the size of $T(N_{G'}(x))$. This is not sufficient to establish an overall linear running time. Indeed, the size of $T(N_{G'}(x))$ can be significantly larger than $|N_{G'}(x)|$. In that case, Lemma 18 says that $|T'(N_{G'}(x))|$ is significantly smaller than $|T(N_{G'}(x))|$. This is the case when a large fully-mixed subtree is contracted into a single prime node. So to circumvent this issue, we show that inserting a good vertex x in a circle graph G takes amortized time $O(|N_{G'}(x)|)$ if $G' = G + x$ is a circle graph. We use the method of potentials introduced in [32]; see [5, Ch. 17.3] for a modern exposition.

We define a potential function Φ on GLT's representing a split decomposition. Let $(T, \mathcal{F})$ be a GLT. For an inner node u of T, we define:

$$\Phi_u = \begin{cases} 1 & \text{if } u \text{ is non-degenerate,} \\ \deg_T(u) - 2 & \text{if } u \text{ is degenerate.} \end{cases}$$

And we set $\Phi(T, \mathcal{F}) = \sum_{u \in T} \Phi_u$. Note that, by definition, the potential is non-negative. The *amortized cost* to insert vertex x is

$$a(x) = t(x) + \Phi(\mathbf{ST}(G + x)) - \Phi(\mathbf{ST}(G)),$$

where $t(x)$ is the *actual cost*, which is given by Lemma 17, that is $O(|T(N_{G'}(x))|)$.

The intuition behind the definition of this potential function is the following. When a large fully-mixed subtree into contracted into a single node, which is a costly operation, then the potential will significantly decrease. By contrast, when a new degenerate node or new leaf adjacent to a degenerate node is added, which can be done efficiently, then the potential increases, but only slightly. This implies the overall amortized linear running time.

▶ **Lemma 19.** *Let $G' = G + x$ be a graph such that x is a good vertex of G and G is a circle graph. If G' is a circle graph, then $\mathbf{PC}(G')$ can be computed from $\mathbf{PC}(G)$ in amortized $O(|N_{G'}(x)|)$ time.*

Proof. Let us denote $\mathbf{ST}(G) = (T, \mathcal{F})$ and $\mathbf{ST}(G') = (T', \mathcal{F}')$. By Lemma 17, the actual cost $t(x)$ is in $O(|T(N_{G'}(x))|)$. We now consider the four cases identified in Subsection 5.1 and in each of them determine the potential difference and the resulting amortized running time.

1. As $\mathbf{ST}(G')$ is obtained from $\mathbf{ST}(G)$ by either adding a chord to a prime node or by adding a new leaf, we have $\Phi(\mathbf{ST}(G')) - \Phi(\mathbf{ST}(G)) \leq 1$. Moreover, we have $|T(N_{G'}(x))| = |T'(N_{G'}(x))|$. So by Lemma 18, $|T(N_{G'}(x))| \leq 2 \cdot |N_{G'}(x)|$. Thus, $t(x)$ is in $O(|N_{G'}(x)|)$ and $\Phi(\mathbf{ST}(G + x)) - \Phi(\mathbf{ST}(G))$ is constant, which yields an amortized running time $a(x)$ in $O(|N_{G'}(x)|)$.
2. In this case $\mathbf{ST}(G')$ is obtained from $\mathbf{ST}(G)$ by subdividing an edge with a new degenerate node of degree 3. Thus we have $\Phi(\mathbf{ST}(G')) - \Phi(\mathbf{ST}(G)) = 1$ and $|T(N_{G'}(x))| = |T'(N_{G'}(x))| - 1$. So by Lemma 18, $|T(N_{G'}(x))| = |T'(N_{G'}(x))| - 1 \leq 2 \cdot |N_{G'}(x)| - 1$. Thus, $t(x)$ is in $O(|N_{G'}(x)|)$ and $\Phi(\mathbf{ST}(G + x)) - \Phi(\mathbf{ST}(G))$ is constant, which yields an amortized running time $a(x)$ in $O(|N_{G'}(x)|)$.
3. In this case $\mathbf{ST}(G')$ is obtained from $\mathbf{ST}(G)$ by splitting a hybrid node u that is degenerate into two adjacent nodes v, w and then using the previous case to handle the edge vw. Observe that performing a node-split on a degenerate node does not change the potential. Moreover as discussed in the previous case, subdividing a tree-edge to insert a degenerate node of degree 3 increases the potential of the resulting GLT by 1. It follows that $\Phi(\mathbf{ST}(G')) - \Phi(\mathbf{ST}(G)) = 1$. Observe that $|T(N_{G'}(x))| = |T'(N_{G'}(x))| - 2$, then by Lemma 18, $|T(N_{G'}(x))| = |T'(N_{G'}(x))| - 2 \leq 2 \cdot |N_{G'}(x)| - 2$. Thus, $t(x)$ is in $O(|N_{G'}(x)|)$ and $\Phi(\mathbf{ST}(G + x)) - \Phi(\mathbf{ST}(G))$ is constant, which yields an amortized running time $a(x)$ in $O(|N_{G'}(x)|)$.

4. Suppose that the fully-mixed subtree of $\mathbf{ST}(G)$ contains d degenerate nodes and p prime nodes. Since each node-split of a degenerate node leaves the potential unchanged, we have $\Phi(\mathsf{cl}(\mathbf{ST}(G)) = \Phi(\mathbf{ST}(G))$. Observe that in $\mathsf{cl}(\mathbf{ST}(G))$, every degenerate node u has degree 3, implying that $\Phi_u = 1$. Thus every node (degenerate or prime) of the fully mixed subtree of $\mathsf{cl}(\mathbf{ST}(G))$ contributes 1 to $\Phi(\mathsf{cl}(\mathbf{ST}(G)))$. Since $\mathbf{ST}(G')$ is obtained by contracting the fully-mixed subtree of $\mathsf{cl}(\mathbf{ST}(G))$ to a single prime node, it follows that $\Phi(\mathbf{ST}(G')) = \Phi(\mathsf{cl}(\mathbf{ST}(G))) - d - p + 1$. So we have $\Phi(\mathbf{ST}(G')) - \Phi(\mathbf{ST}(G)) = -d - p + 1$. Let us now analyze the size of $T'(N_{G'}(x))$ compared to $T(N_{G'}(x))$. Observe that a degenerate node may be split twice, but then in the contraction step, one of the up to three resulting nodes disappears in $T'(N_{G'}(x))$. Moreover every prime node of $T(N_{G'}(x))$ also disappears during the contraction step. It follows that $|T(N_{G'}(x))| \leq |T'(N_{G'}(x))| + d + p - 1$. Then, by Lemma 18, $|T(N_{G'}(x))| \leq 2 \cdot |N_{G'}(x)| + d + p - 1$. It follows that the amortized running time is $a(x) = |T(N_{G'}(x))| + \Phi(\mathbf{ST}(G')) - \Phi(\mathbf{ST}(G)) \leq 2 \cdot |N_{G'}(x)|$. ◀

▶ **Theorem 20.** *Let G be a graph on n vertices and m edges. Deciding if G is a circle graph can be done in $O(n + m)$.*

Proof. Since the correctness of the algorithm is proved in Gioan *et al.* [19], we only discuss the complexity analysis. If G is a circle graph, then by Lemma 19, the algorithm builds the split PC-tree $\mathbf{PC}(G)$ in amortized linear time. So suppose that G is not a circle graph. Let σ be the LexBFS ordering in which the vertices are processed by the algorithm and x be the earliest vertex in σ and S be the subset of vertices containing all vertices y such that $y <_\sigma x$. Then $G[S]$ is a circle graph and we note $\mathbf{ST}(G[S]) = (T_S, \mathcal{F}_S)$. The fact that $G[S] + x$ is not a circle graph is detected either during the computation of $T_S(N(x))$ or when we try to insert the chord corresponding to x to a chord diagram representing a prime node in the split PC-tree in hand. By Lemma 15 and by Lemma 17, this can be decided in time $O(|T_S(N(x))|)$ which is compatible with the amortized complexity analysis of Lemma 19. ◀

References

1 A. Bouchet. Caractérisation des symboles croisés de genre nul. *Compte-rendus Académie des Sciences (CRAS)*, 274:724–727, 1972.

2 A. Bouchet. Reducing prime graphs and recognizing circle graphs. *Combinatorica*, 7:243–254, 1987. `doi:10.1007/BF02579301`.

3 A. Bouchet. Circle graph obstructions. *Journal of Combinatorial Theory Series B*, 60:107–144, 1994. `doi:10.1006/jctb.1994.1008`.

4 G. Brückner, I. Rutter, and P. Stumpf. Extending partial representation of circle graphs in near-linear time. *Algorithmica*, 86(7):2152–2173, 2024. `doi:10.1007/s00453-024-01216-5`.

5 T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 3rd edition, 2001.

6 D.G. Corneil, S. Olariu, and L. Stewart. The LBFS structure and recognition of interval graphs. *SIAM Journal on Computing*, 23(4):1905–1953, 2009. `doi:10.1137/S0895480100373455`.

7 B. Courcelle. Circle graphs and monadic second-order logic. *Journal of Applied Logic*, 6:416–442, 2008. `doi:10.1016/j.jal.2007.05.001`.

8 W.H. Cunningham and J. Edmonds. A combinatorial decomposition theory. *Canadian Journal of Mathematics*, 32(3):734–765, 1980. `doi:10.4153/CJM-1980-057-7`.

9 E. Dahlhaus. Efficient parallel and linear time sequential split decomposition (extended abstract). In *Foundations of Software Technology and Theoretical Computer Science - FSTTCS*, volume 880 of *Lecture Notes in Computer Science*, pages 171–180, 1994. `doi:10.1007/3-540-58715-2_123`.

10 E. Dahlhaus. Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition. *Journal of Algorithms*, 36(2):205–240, 2000. doi:10.1006/jagm.2000.1090.

11 H. de Fraysseix. A characterization of circle graphs. *European Journal of Combinatorics*, 5:223–238, 1984. doi:10.1016/S0195-6698(84)80005-0.

12 S. Even and A. Itai. Queues, stacks and graphs. *Theory of Machines and Computations*, pages 71–86, 1971. doi:10.1016/B978-0-12-417750-5.50011-7.

13 S.D. Fink, M. Pfretzschner, and I. Rutter. Experimental comparison of PC-trees and PQ-trees. *ACM Journal of Experimental Algorithms*, 28:24, 2023. doi:10.1145/3611653.

14 J.C. Fournier. Une caractérisation des graphes de cordes. *Compte-rendus Académie des Sciences (CRAS)*, 286A:811–813, 1978.

15 C.P. Gabor, W.L. Hsu, and K.J. Suppovit. Recognizing circle graphs in polynomial time. *Journal of ACM*, 36:435–473, 1989. doi:10.1145/65950.65951.

16 B.A. Galler and M.J. Fischer. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, 1964. doi:10.1145/364099.364331.

17 J. Geelen, O j. Kwon, R. McCarty, and P. Wollan. The grid theorem for vertex-minors. *Journal of Combinatorial Theory, Series B*, 158:93–116, 2023. doi:10.1016/j.jctb.2020.08.004.

18 E. Gioan and C. Paul. Split decomposition and graph-labelled trees: characterizations and fully dynamic algorithms for totally decomposable graphs. *Discrete Applied Mathematics*, 160:708–723, 2012. doi:10.1016/j.dam.2011.05.007.

19 E. Gioan, C. Paul, M. Tedder, and D.G Corneil. Practical and efficient circle graph recognition. *Algorithmica*, 69:759–788, 2014. doi:10.1007/s00453-013-9745-8.

20 E. Gioan, C. Paul, M. Tedder, and D.G Corneil. Practical and efficient split decomposition via graph-labelled trees. *Algorithmica*, 69:789–843, 2014. doi:10.1007/s00453-013-9752-9.

21 W.-L. Hsu and R. McConnell. PC trees and cicular-ones arrangements. *Theoretical Computer Science*, 296:99–116, 2003. doi:10.1016/S0304-3975(02)00435-8.

22 W.L. Hsu. $O(m \cdot n)$ algorithms for the recognition and isomorphism problems on circular-arc. *SIAM Journal on Computing*, 24(3):411–439, 1995. doi:10.1137/S0097539793260726.

23 V. Kalisz, P. Klavík, and P. Zeman. Circle graph isomorphism in almost linear time. In *Annual Conference on Theory and Applications of Models of Computation - TAMC*, volume 13571 of *Lecture Notes in Computer Science*, pages 176–188, 2022. doi:10.1007/978-3-031-20350-3_15.

24 A. Kotzig. Quelques remarques sur les transformations K. In *Séminaire de Paris*, 1977.

25 T.-H. Ma and J. Spinrad. An $O(n^2)$ algorithm for undirected split decomposition. *Journal of Algorithms*, 16:145–160, 1994. doi:10.1006/jagm.1994.1007.

26 W. Naji. Reconnaissance des graphes de cordes. *Discrete Mathematics*, 54:329–337, 1985. doi:10.1016/0012-365X(85)90117-7.

27 S.I. Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory Series B*, 95:79–100, 2005. doi:10.1016/j.jctb.2005.03.003.

28 N. Robertson and P. Seymour. Graph minors V: excluding a planar graph. *Journal of Combinatorial Theory Series B*, 41:92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

29 N. Robertson and P. Seymour. Graph minors IV. tree-width and well-quasi-ordering. *Journal of Combinatorial Theory Series B*, 48:227–254, 1990. doi:10.1016/0095-8956(90)90120-O.

30 D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. doi:10.1137/0205021.

31 W. K. Shih and W. L. Hsu. A new planarity test. *Theoretical Computer Science*, 223:179–191, 1999. doi:10.1016/S0304-3975(98)00120-0.

32 D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

33 J. Spinrad. Recognition of circle graphs. *Journal of Algorithms*, 16:264–282, 1994. doi:10.1006/jagm.1994.1012.

34 R. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22:215–225, 1975. doi:10.1145/321879.321884.