Algorithmica
https://doi.org/10.1007/s00453-024-01216-5

# Extending Partial Representations of Circle Graphs in Near-Linear Time

Guido Brückner[2] · Ignaz Rutter[3] · Peter Stumpf[1,3]

Received: 17 May 2023 / Accepted: 7 February 2024

## Abstract

The *partial representation extension problem* generalizes the recognition problem for geometric intersection graphs. The input consists of a graph $G$, a subgraph $H \subseteq G$ and a representation $\mathcal{R}'$ of $H$. The question is whether $G$ admits a representation $\mathcal{R}$ whose restriction to $H$ is $\mathcal{R}'$. We study this question for *circle graphs*, which are intersection graphs of chords of a circle. Their representations are called *chord diagrams*. We show that for a graph with $n$ vertices and $m$ edges the partial representation extension problem can be solved in $O((n+m)\alpha(n+m))$ time, thereby improving over an $O(n^3)$-time algorithm by Chaplick et al. (J Graph Theory 91(4), 365–394, 2019). The main technical contributions are a canonical way of orienting chord diagrams and a novel compact representation of the set of all canonically oriented chord diagrams that represent a given circle graph $G$, which is of independent interest.

**Keywords** Circle graphs · Simultaneous representation · Geometric intersection graphs · Recognition

Guido Brückner, Ignaz Rutter and Peter Stumpf have contributed equally to this work.

✉ Peter Stumpf
stumpf@kam.mff.cuni.cz

Guido Brückner
guido.brueckner@gmail.com

Ignaz Rutter
rutter@fim.uni-passau.de

1 Department of Applied Mathematics (KAM), Charles University, Prague, Czech Republic

2 Department of Informatics, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

3 Faculty of Computer Science and Mathematics, University of Passau, Passau, Germany

## 1 Introduction

Geometric intersection representations of graphs are an important concept that establishes a strong tie between geometry, combinatorics, and graph theory. In an intersection representation of a graph $G = (V, E)$ each vertex $v \in V$ is represented by a geometric object $R(v)$ whose intersections encode the edges of $G$, i.e., $\{u, v\} \in E$ if and only if $R(u)$ and $R(v)$ intersect. Different classes of graphs can be obtained by restricting the types of geometric objects used for the representation. For example *interval graphs* are intersection representations of intervals of the real line, *string graphs* are intersection graphs of curves in the plane and *circle graphs* are intersection graphs of chords of a circle.

For a fixed class $\mathcal{C}$ of intersection graphs, a natural question is the *recognition problem*, which asks whether a given $G$ belongs to $\mathcal{C}$. For circle graphs the recognition problem has been studied for a long time, and has culminated in an algorithm with running time $O((n+m)\alpha(n+m))$ [1], where $n$ and $m$ denote the number of vertices and edges of the input graph, respectively, and $\alpha$ denotes the slowly growing inverse of the Ackermann function. There is also an $O(n^2)$ time algorithm which is faster for very dense graphs [2].

A generalization of the basic recognition problem has attracted considerable attention: the *partial representation extension problem*. In the partial representation extension problem, the input consists of a triplet $(G, H, \mathcal{R}')$, where $G$ is a graph, $H \subseteq G$ is an induced subgraph of $G$, and $\mathcal{R}'$ is an intersection representation of $H$. The question is whether there exists a representation $\mathcal{R}$ of $G$ that *extends* $\mathcal{R}'$, i.e., its restriction to $H$ coincides with $\mathcal{R}'$. Note that for such an extension to exist, it is necessary that $H$ is an induced subgraph of $G$. Following the notation of Chaplick, Fulek and Klavík [3], for a class $\mathcal{C}$ of intersection graphs, we denote the partial representation extension problem for $\mathcal{C}$ by $\textsc{RepExt}(\mathcal{C})$.

**Related Work.** The recognition problem can be solved efficiently for a wide range of classes of intersection graphs. The partial representation extension problem was introduced by Klavík et al. [4], who gave an efficient algorithm for $\textsc{RepExt}(\textsc{Int})$, where $\textsc{Int}$ denotes the class of interval graphs, which they improved to linear running time in their full version [5]. Angelini et al. [6] give a linear-time algorithm for planar topological graph drawings and Patrignani shows NP-hardness for planar straight-line drawings [7]. Recently, the problem has also been considered for simple topological and 1-planar drawings [8, 9]. In the meantime efficient algorithms are known for proper and unit interval graphs [10], permutation graphs and function graphs [11], as well as for trapezoid graphs [12]. Concerning the class $\textsc{Circle}$ of circle graphs, Chaplick et al. [3] gave the first efficient algorithm for $\textsc{RepExt}(\textsc{Circle})$ with running time $O(n^3)$.

For other forms of graph representations, there are compact descriptions of all representations of a graph $G$, e.g. SPQR-trees [13] for planar graphs and modular decomposition trees [14] for comparability graphs. Both descriptions express a representation of $G$ by choosing representations for small graphs that are associated to the nodes of a tree in such a way that a bijection is obtained between the representations of $G$ and these choices. For circle graphs, Gioan et al. introduced split-trees [15] that decompose a graph along its splits into smaller graphs from which $G$ is assembled in a tree structure. They observe that all possible chord diagrams of $G$ can be obtained by choosing a chord diagram for each of the small graphs associated to the nodes of the split-tree of $G$ and combining them suitably with each other. This has the downside that different choices can lead to the same chord diagram of $G$.

**Contribution and Outline.** We strengthen this connection by introducing a canonical orientation of chord diagrams so that we indeed obtain a bijection between the canonically oriented chord diagrams of $G$ and choices of canonically oriented chord diagrams for the nodes of the (canonical) split-tree of Gioan et al. We call such a choice for each node a *configuration*. We develop the theory and linear-time algorithms for transforming a canonically oriented chord diagram to a corresponding configuration and vice versa.

We emphasize the importance of the properties that (i) the correspondence between configurations and diagrams is a bijection and (ii) the representation is canonical, i.e., it depends only on the choice of a single root vertex. The former is essential, as it allows us to compare diagrams by comparing configurations. The latter is crucial since in our applications we sometimes work with different split-trees (non-canonical split-trees can be used to represent subsets of all representations, e.g., ruling out some that do certainly not extend a given partial representation). The orientations are used to determine the mapping from configurations to diagrams, i.e., the way the representation combines the diagrams of the graphs associated with the individual nodes of the tree. If this information were specified by additional, external information, it would not be possible to stably compare configurations across different split-trees of the same graph in such a way that equal configurations imply equal diagrams. Due to the canonical orientations, it suffices to root the split-trees at the same vertex.

To illustrate the usefulness of our representation, we show how to extend partial chord diagrams in near-linear time $O((n+m)\alpha(n+m))$, improving over the $O(n^3)$-time algorithm of Chaplick et al. [3]. In fact, Chaplick et al. [3] and also Kalisz et al. [16] ask specifically whether such a representation of all chord diagrams exists and whether it can be used to solve RepExt(Circle) faster as open questions. We note that, given the data structure computed by the fastest known recognition algorithm [1] for $G$, our algorithm runs in linear time.

We introduce notation and preliminary definitions in Sect. 2. In Sect. 3 we develop the compact description of representations. Section 4 shows how these results can be used to obtain an almost-linear time algorithm for RepExt(Circle).

## 2 Preliminaries

In this section we introduce important concepts that we use throughout the paper. In particular, we recall the concepts of circle graphs and chord diagrams, and we introduce a way to canonically orient them. Moreover, we also recall the notion of splits and split decompositions, which are a classic tool in circle graph recognition algorithms.

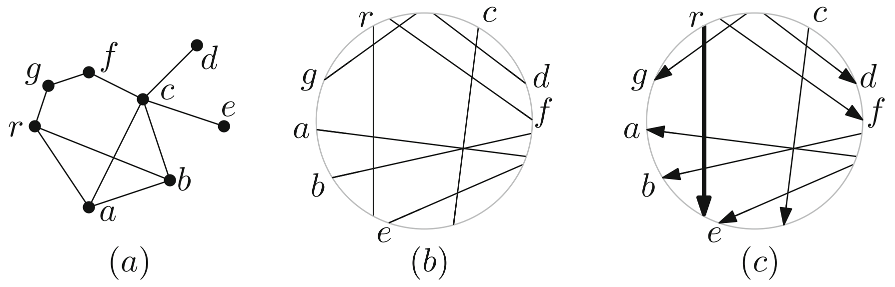

**Fig. 1** **a** A graph $G$ **b** an undirected chord diagram of $G$ **c** an oriented chord diagram of $G$ with reference chord $r$ and starting point at the top

## 2.1 Circle Graphs and Chord Diagrams

An *(undirected) chord diagram* $D$ consists of a set $C$ of *chords* of the unit circle, i.e., undirected straight-line segments that connect pairwise distinct points on the unit circle. A chord diagram $D$ naturally defines an intersection graph $G(D) = (C, E)$ of its chords, where $\{c, c'\} \in E$ if and only if the chords $c$ and $c'$ intersect in $D$, i.e., if and only if their endpoints alternate around the unit circle. A graph $G$ is a *circle graph* if it is an intersection graph of the chords of a chord diagram (Fig. 1).

While chord diagrams are geometric objects, we are most interested in their combinatorial structure. To break certain symmetries we usually consider *oriented* chord diagrams, which additionally have a chord end as a *starting point*. Then an oriented chord diagram $D$ can be encoded as the word over the set of chords $C$ obtained by starting at the starting point and walking around the unit circle in clockwise direction and recording the encountered chords. For $c \in C$, we refer to the first end of $c$ that we encounter with $\mathring{c}$ and to the second end with $\hat{c}$. We consider the chord $c$ as directed from $\mathring{c}$ to $\hat{c}$ in the geometric interpretation; see Fig. 2 (reference). Unless mentioned otherwise, chord diagrams are always considered oriented and they are usually treated as their encodings.

We call the chord with the starting point the *reference chord* of $D$. We never change the reference chord. In the following, we consider changes that can be applied to every chord diagram without changing the represented graph or the reference chord. For a word $w$ we write $w^{\text{rev}}$ for the word obtained by reading $w$ backwards. The *reverse* of a chord diagram $a\rho a\sigma$ is $\text{rev}(a\rho a\sigma) = a\sigma^{\text{rev}}a\rho^{\text{rev}}$. Geometrically, this corresponds to mirroring the chord diagram at the reference chord $a$; see Fig. 2 (reversed). The *turn* of a chord diagram $a\rho a\sigma$ is $\text{turn}(a\rho a\sigma) = a\sigma a\rho$. Geometrically, this corresponds to choosing $\hat{a}$ as the new starting point. In a sense, this turns the chord diagram by 180 degrees; see Fig. 2 (turned). Note that $\text{turn}(\text{turn}(a\rho a\sigma)) = a\rho a\sigma$, $\text{rev}(\text{rev}(a\rho a\sigma)) = a\rho a\sigma$ and $\text{turn}(\text{rev}(a\rho a\sigma)) = \text{turn}(a\sigma^{\text{rev}}a\rho^{\text{rev}}) = a\rho^{\text{rev}}a\sigma^{\text{rev}} = \text{rev}(a\sigma a\rho) = \text{rev}(\text{turn}(a\rho a\sigma))$, i.e., turn and rev are selfinverse and commute.

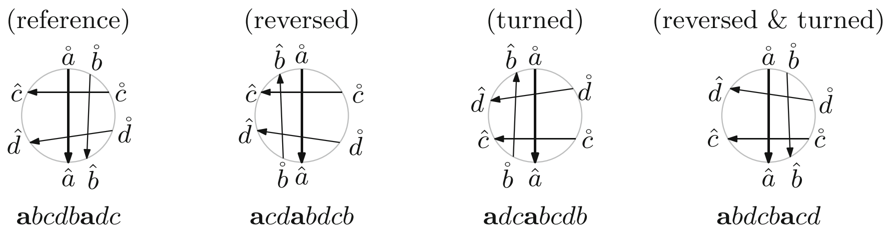

**Fig. 2** The turn and rev operation on a chord diagram with reference chord $a$

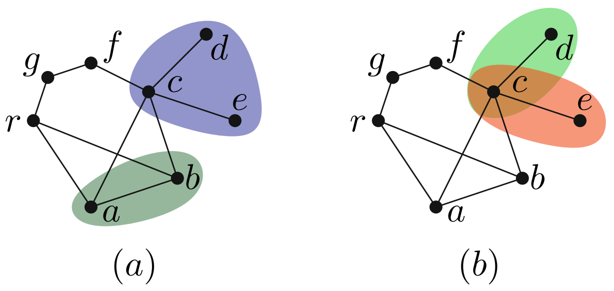

**Fig. 3** **a** split $(\{a, b\}, \{c, d, e, f, g, r\})$ with boundary $(\{a, b\}, \{c, r\})$, split $(\{c, d, e\}, \{a, b, f, g, r\})$ with boundary $(\{c\}, \{a, b, f\})$ **b** split $(\{c, d\}, \{a, b, e, f, g, r\})$ with boundary $(\{c\}, \{a, b, e, f\})$ and split $(\{c, e\}, \{a, b, d, f, g, r\})$ with boundary $(\{c\}, \{a, b, d, f\})$. The splits in **a** are good while the splits in **b** overlap and are not good

## 2.2 Splits and Split Trees

Let $G = (V, E)$ be a graph. Consider a bipartition $(V_A, V_B)$ of $V$ with $\emptyset \subsetneq V_A, V_B \subsetneq V$ and let $A \subseteq V_A$ denote the subset of vertices of $V_A$ that are adjacent to a vertex in $V_B$ and let $B \subseteq V_B$ denote the subset of vertices of $V_B$ that are adjacent to a vertex in $V_A$. The bipartition $(V_A, V_B)$ is called a *split* if all possible edges between $A$ and $B$ exist in $G$, i.e., $\{\{a, b\} \mid a \in A, b \in B\} \subseteq E$. Then $(A, B)$ is called the *split boundary*; see Fig. 3. Observe that if $V_A$ and $V_B$ are not connected, then $A$, $B$ are empty. A split $(V_A, V_B)$ is *trivial* if one among $V_A$ and $V_B$ consists of a single vertex. Following Courcelle [17], we call a split $(V_A, V_B)$ *good* if it does not overlap any other split, in the sense that for any other split $(V_C, V_D)$ at least one of $V_A \cap V_C$, $V_A \cap V_D$, $V_B \cap V_C$, $V_B \cap V_D$ is empty. A graph is *prime* if all its splits are trivial, and it is *degenerate* if every bipartition of the vertices yields a split. It is well known that the connected degenerate graphs are precisely cliques and stars [18]. An example for a prime graph is $C_n$, the cycle on $n$ vertices, for any $n \geq 5$. Suppose there was a non-trivial split $(V_A, V_B)$ in $C_n$. Since $C_n$ is 2-connected, both $A$ and $B$ contain at least two vertices each. This implies the complete bipartite graph $K_{2,2}$ between $A$ and $B$ (with two vertices on each side), which is not subgraph of $C_n$, a contradiction. Hence, $C_n$ is prime for $n \geq 5$. Bouchet [19] showed that the undirected chord diagram of a connected prime circle graph is unique up to reversal.

Next, we define *split trees* as introduced by Gioan et al. [15]. To avoid confusion with the vertices and edges of our graphs, we refer to the vertices and edges of split

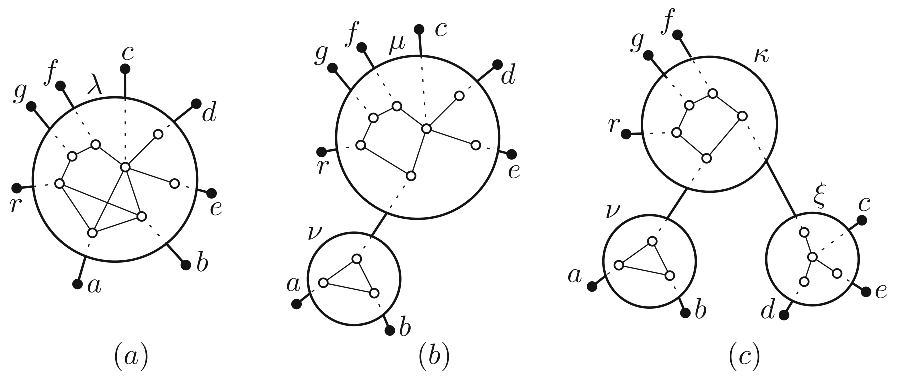

**Fig. 4** Split trees of the graph from Fig. 1a. **a** shows the base case, in **b** the node $\lambda$ from (**a**) is decomposed into $\mu$ and $\nu$ along the split $(\{a, b\}, \{c, d, e, r, g, f\})$. In (**c**) the node $\mu$ from (**b**) is further decomposed along the split that has $\{c, d, e\}$ on one side

trees as nodes and arcs, respectively. A split tree $T$ is a tree where each inner node $\mu$ has a *skeleton graph* $\mathrm{skel}(\mu)$ and a bijection $\mathrm{corr}_\mu$ from $V(\mathrm{skel}(\mu))$ to the nodes of $T$ adjacent to $\mu$; see Fig. 4. Given an inner node $\mu$ and a neighbor $\nu$ of $\mu$, we often need to refer to the vertex $v$ of $\mathrm{skel}(\mu)$ that represents $\nu$. For convenience, we define $v_\mu(\nu)$ as the vertex $v$ of $\mathrm{skel}(\mu)$ with $\mathrm{corr}_\mu(v) = \nu$.

Let $G = (V, E)$ be a graph. In the base case, see Fig. 4a, a *split tree $T$ of $G$* consists of one inner node $\lambda$ and one leaf for each vertex $v \in V$ that is adjacent to $\lambda$. We identify the leaves of $T$ with the vertices in $V$. Define the skeleton of $\lambda$ as $\mathrm{skel}(\lambda) = G$. For every vertex $v \in V(\mathrm{skel}(\lambda))$, we define $\mathrm{corr}_\lambda(v)$ as the leaf node $v$ of $T$.

We can further decompose such a split tree. Let $\lambda$ be an inner node and let $(V_A, V_B)$ be a non-trivial split of $\mathrm{skel}(\lambda)$. Let $G_A$ denote the graph obtained from $\mathrm{skel}(\lambda)$ by contracting $V_B$ into a single vertex $b$. Symmetrically, let $G_B$ denote the graph obtained from $\mathrm{skel}(\lambda)$ by contracting $V_A$ into a single vertex $a$. Then $\lambda$ can be split into two nodes $\mu, \nu$ connected by an arc $\{\mu, \nu\}$; see Fig. 4b. Define $\mathrm{skel}(\mu) = G_A$ and $\mathrm{skel}(\nu) = G_B$. Observe that $b$ in $G_A$ represents the graph $G_B$ and $a$ in $G_B$ represents the graph $G_A$. We define $\mathrm{corr}_\mu(b) = \nu$ and $\mathrm{corr}_\nu(a) = \mu$. For any vertex $v \in V(\mathrm{skel}(\lambda))$ we replace the arc $\{\lambda, \mathrm{corr}_\lambda(v)\}$ with $\{\mu, \mathrm{corr}_\lambda(v)\}$ if $v \in V_A$ or $\{\nu, \mathrm{corr}_\lambda(v)\}$ if $v \in V_B$. Finally, for each inner node $\kappa$ adjacent to $\lambda$ consider the unique vertex $v \in \mathrm{skel}(\kappa)$ with $\mathrm{corr}_\kappa(v) = \lambda$. We redefine $\mathrm{corr}_\kappa(v) = \mu$ if $v_\lambda(\kappa) \in V_A$ and $\mathrm{corr}_\kappa(v) = \nu$ if $v_\lambda(\kappa) \in V_B$. Observe that the result is still a tree and $\mathrm{corr}_\xi$ is well defined for each inner node $\xi$. Hence, the inner nodes may be decomposed again. The split trees of $G$ are the split trees that can be obtained in this way.

The inverse of a decomposition is a *join*. Let $G_1, G_2$ be two (vertex-disjoint) graphs with $v_2 \in G_1$, $v_1 \in G_2$. Then we define their *join at* $v_2$,$v_1$ as the graph obtained from $G_1 \cup G_2$ by connecting all neighbors of $v_2$ in $G_1$ with all neighbors of $v_1$ in $G_2$ and removing the vertices $v_1, v_2$. We denote the resulting graph by $G_1 \oplus_{v_2,v_1} G_2 = (V, E)$. For a split tree $T$ with an arc $\{\mu, \nu\}$ let $b = v_\mu(\nu)$, $a = v_\nu(\mu)$. The nodes $\mu, \nu$ can be joined into a single node $\lambda$ with $\mathrm{skel}(\lambda) = \mathrm{skel}(\mu) \oplus_{b,a} \mathrm{skel}(\nu)$. Moreover, for any inner node $\kappa$ adjacent to $\mu$ or $\nu$ and the unique vertex $v \in V(\text{skel}(\kappa))$ with $\text{corr}_\kappa(v) \in \{\mu, \nu\}$ we redefine $\text{corr}_\kappa(v) = \lambda$.

Let $T$ denote a split tree and let $\{\mu, \nu\}$ be an arc of $T$. Removing $\{\mu, \nu\}$ separates $T$ into two trees $T_\mu$ and $T_\nu$ where $T_\mu$ contains the node $\mu$ and $T_\nu$ contains the node $\nu$. Let $L(T_\mu) \subseteq V$ or simply $L(\mu)$ denote the set of leaves of $T_\mu$. By construction of the split-tree, $(L(T_\mu), L(T_\nu))$ is a split of $G$, which we call *induced* by the arc $\{\mu, \nu\}$. For example, the blue split from Fig. 3a is induced by the arc $\{\kappa, \xi\}$ in Fig. 4c. Let $(V_A, V_B)$ be a non-trivial split of $\text{skel}(\mu)$. This split *represents* the split $(L_A, L_B)$ of $G$ with $L_A = \bigcup_{v \in V_A} L(T_{\text{corr}_\mu(v)})$ and $L_B = \bigcup_{v \in V_B} L(T_{\text{corr}_\mu(v)})$. An example of this is the red split from Fig. 3b that is represented by a non-trivial split inside node $\xi$ of Fig. 4c. We call an inner node of a split tree *degnerate* and *prime*, if its skeleton graph is degenerate and prime, respectively. Observe that in the split tree in Fig. 4c, the node $\kappa$ is prime, and $\nu$, $\xi$ are degenerate.

In a *good-split tree* every inner arc $\{\mu, \nu\}$ of $T$ induces a good split. For a connected graph $G$, a *canonical split tree* is a good-split tree such that no skeleton has a non-trivial good split, no inner arc induces a trivial split, and any two arcs induce different splits. A canonical split tree is obtained by decomposing $G$ (in arbitrary order) along all its good splits. This is possible since the good splits form a laminar set. The resulting canonical split tree is denoted by $\text{ST}(G)$. It was first defined by Gioan et al. [15], who proved several useful statements about it in the case where $G$ is connected. In particular, it is unique, and nodes have connected skeletons that are either prime or degenerate. Moreover, if two adjacent nodes $\mu$, $\nu$ are both degenerate, then either one of them is a star and one is a clique, or they are both stars and the vertices $v_\mu(\nu)$ and $v_\nu(\mu)$ are either both leaves or both the star centers of their respective skeletons. The following useful property states that $\text{ST}(G)$ represents all splits of $G$.

**Proposition 1** ([18], [15][Theorem 2.18]) *A partition $(V_A, V_B)$ of the vertex set of a connected graph $G$ is a split of $G$ if and only if it is induced by an arc of $ST(G)$ or represented by a non-trivial split of a (skeleton of a) degenerate node of $ST(G)$.*

## 3 Compact Representation of all Chord Diagrams

Let $G = (V, E)$ be a circle graph. In Sect. 3.1 we establish a correspondence between the chord diagrams of $G$ and assignments of chord diagrams to the inner nodes of a split tree of $G$. For canonical split trees, this correspondence turns out to be a bijection; this gives the claimed compact representation for connected circle graphs, answering the open question by Chaplick et al. [3] and Kalisz et al. [16]. We further show that this bijection can be computed in both directions in linear time. In Sect. 3.2 we analyse the possible choices for the chord diagrams. Altogether, this gives the claimed compact representation for connected circle graphs.

### 3.1 Configurations of Split Trees

Let $G = (V, E)$ be a circle graph, let $r \in V$ be the reference chord of $G$ and let $T$ be a split tree of $G$. We root $T$ at $r$ and direct all arcs away from the root. For each inner

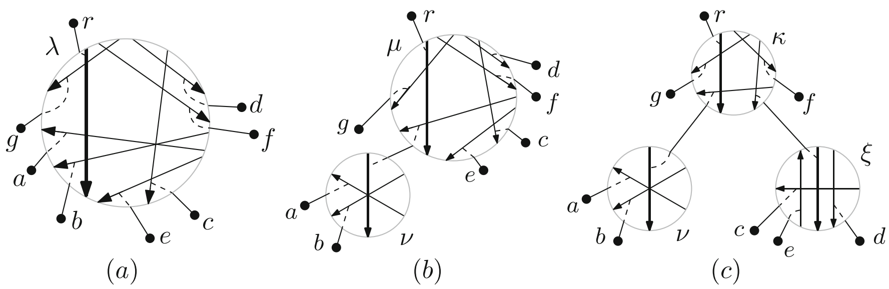

**Fig. 5** **a**, **b**, **c**, show configurations of the corresponding split trees in Fig. 4 that yield the same chord diagram. We have $c'(\lambda) = rfgdcdf\mathbf{ba}ecer\mathbf{ba}g = rfgdcdf\mathbf{v}ecer\mathbf{v}g \oplus_{v,r_\nu} r_\nu bar_\nu ba = c(\mu) \oplus_{v,r_\nu} c(\nu)$, where $v = v_\mu(\nu)$

node $\nu$ of $T$, we define the reference chord $r_\nu$ as the chord $v_\nu(\mu)$ associated with the parent node $\mu$. A *configuration* $c$ of $T$ is a mapping that assigns to each inner node $\mu$ of $T$ a chord diagram $c(\mu)$ of skel$(\mu)$ with reference chord $r_\mu$; see Fig. 5. We also refer to $c(\mu)$ as the *configuration of* $\mu$. Recall that a split decomposition can be used to split a node $\lambda$ of $T$ into two nodes $\mu, \nu$ connected by a (directed) arc $(\mu, \nu)$. In the reverse direction, a join composition can be used to join two nodes $\mu, \nu$ connected by an arc $(\mu, \nu)$ into a single node $\lambda$. The join operation extends to configurations.

The configurations of $\mu$ and $\nu$ induce a unique configuration of $\lambda$ as follows. Let $a = v_\mu(\nu)$ and let $c(\nu) = r_\nu \rho r_\nu \sigma$. The induced configuration $c'(\lambda) = c(\mu) \oplus_{a,r_\nu} c(\nu)$ of $\lambda$ is obtained from $c(\mu)$ by replacing $\mathring{a}$ with $\rho$ and $\hat{a}$ with $\sigma$; see Fig. 5b.

**Lemma 2** *Let $T$ be a split tree of a graph $G$. Then applying a set of joins in two different orders on $T$ results in the same split tree $T'$. Further, given a configuration $c$ for $T$, the resulting configuration of $T'$ is also the same.*

***Proof*** It suffices to prove the statement for a set of two joins. Consider two pairs $\mu, \nu$ and $\lambda, \kappa$ of two vertices in $T$ with $\{\mu, \nu\}, \{\lambda, \kappa\} \in E(T)$, where $\mu, \lambda$ are the parents of $\nu, \kappa$. Let $a = v_\nu(\mu)$, $b = v_\mu(\nu)$,, $e = v_\lambda(\kappa)$, $d = v_\kappa(\lambda)$. Let, in each of the following cases, $\zeta$ be the vertex resulting from joining at $a$,$b$ and let $\eta$ be the vertex resulting from joining at $e, d$.

We consider the results $T_1, c_1$ of joining first at $a, b$ and then at $e, d$ and the results $T_2$, $c_2$ of joining first at $c, d$ and then at $a, b$ (where $T_1, T_2$ are the split-trees and $c_1, c_2$ are the configurations). First assume the four nodes $\mu, \nu, \lambda, \kappa$ are pairwise distinct. Then we have $c(\mu) = \alpha_1 b \alpha_2 b \alpha_3$, $c(\nu) = a\beta_1 a \beta_2$, $c(\lambda) = \delta_1 e \delta_2 e \delta_3$, $c(\kappa) = d\gamma_1 d\gamma_2$. In both, $T_1$ and $T_2$, we obtain skel$(\zeta)$ = skel$(\mu) \oplus_{b,a}$skel$(\nu)$ and skel$(\eta)$ = skel$(\lambda) \oplus_{e,d}$skel$(\kappa)$ as well as $c_1(\zeta) = c_2(\zeta) = \alpha_1\beta_1\alpha_2\beta_2\alpha_3$ and $c_1(\eta) = c_2(\eta) = \delta_1\gamma_1\delta_2\gamma_2\delta_3$.

Next assume $\mu = \lambda$. Then we have $b, d \in c(\mu)$, $c(\nu) = a\beta_1 a\beta_2$, $c(\kappa) = d\gamma_1 d\gamma_2$. For the vertex $\xi$ resulting from both joins, we have that $\mathring{b}, \hat{b}, \mathring{d}, \hat{d}$ have been replaced by $\beta_1, \beta_2, \gamma_1, \gamma_2$ independently in $c_1(\xi)$ and $c_2(\xi)$ which implies $c_1(\xi) = c_2(\xi)$ and thus (skel$(\mu) \oplus_{b,a}$ skel$(\nu)) \oplus_{e,d}$ skel$(\kappa)$ = (skel$(\mu) \oplus_{e,d}$ (skel$(\kappa)) \oplus_{b,a}$ skel$(\nu)$).

Finally assume $\nu = \lambda$. Then we have $c(\mu) = \alpha_1 b\alpha_2 b\alpha_3$, $e \in c(\nu) = a\beta_1 a\beta_2$, $c(\kappa) = d\gamma_1 d\gamma_2$. Let $\beta'_1, \beta'_2$ be the words obtained by replacing $\mathring{c}, \hat{c}$ in $\beta_1, \beta_2$ by $\gamma_1, \gamma_2$. If we first join at $a, b$, then we have $e \in c(\zeta) = \alpha_1\beta_1\alpha_2\beta_2\alpha_3$ and after joining at $e, d$ we obtain $c(\eta) = \alpha_1\beta_1'\alpha_2\beta_2'\alpha_3$. If we first join at $e, d$, then we have $c(\eta) = a\beta_1'a\beta_2'$ and after joining at $a, b$ we obtain $c(\zeta) = \alpha_1\beta_1'\alpha_2\beta_2'\alpha_3$. Therefore, the resulting vertex $\xi$ has in both cases the same assigned chord diagram $c_1(\xi) = c_2(\xi)$ and thus, we obtain $(\text{skel}(\mu) \oplus_{b,a} \text{skel}(\nu)) \oplus_{e,d} \text{skel}(\kappa) = \text{skel}(\mu) \oplus_{b,a} (\text{skel}(\nu) \oplus_{e,d} \text{skel}(\kappa))$. for $\mu = \kappa$ the statement holds by symmetry. □

It follows that any configuration $c$ of a split tree $T$ of a circle graph $G$ uniquely defines a chord diagram $D(c)$ for $G$, which is obtained by joining all internal nodes of $T$ in arbitrary order. For connected graphs, two different configurations yield two different chord diagrams.

**Lemma 3** *Let G be a connected circle graph and let T be a split tree. Then the mapping D that maps configurations of T to chord diagrams of G is injective.*

***Proof*** The proof is by induction on the number $k$ of internal nodes of the split tree. If $k = 1$, then the skeleton of the sole internal node $\mu$ is isomorphic to $G$. Therefore a configuration directly corresponds to a chord diagram of $G$, and the statement holds trivially.

Now assume $k > 1$ and let $c_1, c_2$ be distinct configurations of $T$. Since $c_1 \neq c_2$ there is an internal node $\mu$ of $T$ such that $c_1(\mu) \neq c_2(\mu)$. Since $k > 1$, $\mu$ has an adjacent internal node $\nu$. We join $\mu$ and $\nu$ into a single node $\lambda$ and we let $c_1'$ and $c_2'$ be the corresponding joined configurations on the resulting tree $T'$. If $c_1'$ and $c_2'$ are different, then by the inductive hypothesis, we have that $D(c_1) = D(c_1') \neq D(c_2') = D(c_2)$. It suffices to show $c_1'(\lambda) \neq c_2'(\lambda)$.

If $\nu$ is the parent of $\mu$, then let $v = v_\nu(\mu)$ and let $c_1(\mu) = a\beta_1a\beta_2, c_2(\mu) = a\beta_1'a\beta_2'$. This implies $\beta_1 \neq \beta_1'$ or $\beta_2 \neq \beta_2'$. We obtain $c_1'(\lambda), c_2'(\lambda)$ by replacing $\mathring{v}, \hat{v}$ by $\beta_1, \beta_2$ or $\beta_1', \beta_2'$, respectively. Since skel$(\lambda)$ is connected, there is at least one chord end between $\mathring{v}, \hat{v}$ and we obtain $c_1'(\lambda) \neq c_2'(\lambda)$.

Otherwise, $\mu$ is the parent of $\nu$. Let $u = v_\mu(\nu)$ and let $c_1(\nu) = b\alpha_1b\alpha_2, c_2(\mu) = b\alpha_1'b\alpha_2'$. Since $c_1(\mu) \neq c_2(\mu)$, there are two chords $x, y$ with different order in $c_1(\mu)$, $c_2(\mu)$. If $x, y \neq u$, then they also have different order in $c_1'(\lambda), c_2'(\lambda)$. If $x = u$, note that $V(\nu)$, $V(\mu)$ are disjoint and $\alpha, \alpha_1'$ are not empty. Thus there is a chord in $V(\nu)$ before $y$ in $c_1'(\lambda)$ if and only if $\mathring{u}$ comes before $\mathring{y}$ in $c_1(\mu)$. The same holds for $c_2', c_2$ which implies $c_1'(\lambda) \neq c_2'(\lambda)$. □

In general, not every chord diagram $D$ of $G$ can be obtained from a configuration of $T$ since $D$ cannot necessarily be decomposed along an arbitrary split $(V_A, V_B)$. If $G$ is connected, a diagram $D$ can be decomposed only if the endpoints of the chords in $V_A$ and $V_B$ appear suitably in $D$. We say that $D$ *respects* the split $(V_A, V_B)$ if we have $D = \rho_1\sigma_1\rho_2\sigma_2\rho_3$ where $\rho_2$, $(\rho_3\rho_1)$ are words over $V_A$ and $\sigma_1, \sigma_2$ are words over $V_B$.

Let $D$ be a chord diagram of $G$. Let $T$ denote the split tree that consists of a single inner node $\lambda$ with skel$(\lambda) = G$ and a configuration $c$ with $c(\lambda) = D$. Now consider a split $(V_A, V_B)$ of skel$(\lambda)$ that is respected by $D$. This split decomposes $\lambda$ into two nodes $\mu, \nu$ such that—without loss of generality—$\mu$ is the parent of $\nu$. Let further $a = v_\mu(\nu)$. Because $c(\lambda)$ respects $(V_A, V_B)$, it is of the form $c(\lambda) = \rho_1\sigma_1\rho_2\sigma_2\rho_3$ such that $\rho_1, \rho_2, \rho_3$ contain only chords in $V_A$ and similarly, $\sigma_1, \sigma_2$ contain only chords in $V_B$. We define the configurations of $\mu, \nu$ by setting $c(\mu) = \rho_1 a \rho_2 a \rho_3$ and $c(\nu) = r_\nu \sigma_1 r_\nu \sigma_2$. The reference chord $r_\lambda$ is the reference chord of skel$(\mu)$ and $r_\nu$ is the reference chord of skel$(\nu)$.

Observe that splitting a chord diagram along such a split and joining the resulting two chord diagrams are inverse operations. Let $c(\lambda)$ respect another split$(V'_A, V'_B)$ with $V'_A \subseteq V_A$. Then $(V'_A, V_A \backslash V'_A \cup \{a\})$ is a split in skel$(\mu)$ and $c(\mu)$ respects it. A similar observation holds for splits $(V'_A, V'_B)$ with $V'_B \supseteq V_B$ and $c(\nu)$. Therefore we can recursively decompose a chord diagram along several splits if it respects each of them.

**Lemma 4** *Let $G$ be a (not necessarily connected) circle graph and let $T$ be a split tree of $G$ rooted at some reference chord $r$. Then the mapping $D$ that maps configurations of $T$ to chord diagrams of $G$ is surjective on the set of chord diagrams of $G$ with reference chord $r$ that respect all splits induced by arcs of $T$.*

***Proof*** We use induction on the number $k$ of internal nodes of $T$. If $k = 1$, then skel$(\mu)$ for the sole internal node $\mu$ of $T$ is isomorphic to $G$, and the claim holds.

Assume $k > 1$. Let $D$ be a chord diagram of $G$ with reference chord $r$ that respects all splits induced by arcs of $T$. Let $T'$ be the split tree obtained from $T$ by joining two adjacent internal nodes $\mu, \nu$ into a single node $\lambda$. Clearly $T'$ is a split tree with $k - 1$ internal nodes, and by the inductive hypothesis there exists a configuration $c'$ of $T'$ with $D(c') = D$. Because $D$ respects the split induced by $\{\mu, \nu\}$, $c'(\lambda)$ respects the corresponding split in skel$(\lambda)$. Therefore $c'(\lambda)$ can be split into configurations $c(\mu), c(\nu)$. For all other nodes $\kappa$ of $T$ we set $c(\kappa) = c'(\kappa)$. Note that first joining the nodes $\mu$ and $\nu$ yields the configuration $c'$, and therefore $D(c) = D(c') = D$. □

Note that, for any configuration $c$ of $T$, the chord diagram $D(c)$ respects all splits induced by arcs of $T$. For good splits the situation is particularly simple.

**Proposition 5** ([17][Proposition 9]) *Let $G = (V, E)$ be a connected circle graph, let $D$ be a chord diagram of $G$. Then $D$ respects every good split of $G$.*

We can conclude the following.

**Theorem 6** *Let $T$ be a good-split tree of a connected circle graph $G$ rooted at some reference chord $r \in V(G)$. Then the mapping $D$ that maps configurations of $T$ to chord diagrams of $G$ with reference chord $r$ is a bijection.*

***Proof*** Direct consequence of Lemmas 3 and 4 with Proposition 5. □

We can translate between configuration and chord diagram in linear time, which allows us to use this result algorithmically.

**Theorem 7** *Let $T$ be a split tree of a connected circle graph $G$ rooted at some reference chord $r \in V(G)$. Then the mapping $D$ that maps configurations of $T$ to chord diagrams of $G$ can be computed in linear time. Conversely, given a chord diagram $D$ of $G$ with reference chord $r$, it can be tested in linear time whether there exists a configuration $c$ of $T$ with $D(c) = D$. If it exists, the configuration $c$ can also be computed in linear time.*

***Proof*** We store chord diagrams as circular doubly linked lists of endpoints of chords in clockwise order. We assume that each chord endpoint is equipped with a pointer to the corresponding vertex and each vertex has pointers to the two endpoints of its chord.

Let $c$ be a configuration of $T$. We first store for each chord $u$ in a chord diagram which of its ends is $\mathring{u}$. We then process the tree $T$ in bottom-up order. If $c$ contains only a single inner node, then the configuration of this node is the desired diagram $D(c)$. Otherwise let $\mu$ be an inner node of $T$ whose children are all leaves and let $\nu$ be its parent. Let $T'$ be the split tree obtained by replacing in $T$ the node $\mu$ together with all its leaves by a single leaf $v$ and let $c'$ be the configuration of $T'$ that coincides with $c$ for all nodes of $T'$. Clearly, $c'$ and $T'$ can be computed from $c$ and $T$ in $O(1)$ time. We now recursively compute $D(c')$ in time linear in the size of $T'$. To obtain $D(c)$, we replace in $D(c')$ the endpoints $\mathring{v}$, $\hat{v}$ of $v$ by the sequences $\pi$, $\sigma$, respectively, where $c(\mu) = r\pi r\sigma$ and $r$ is the reference chord of $\mu$. Since we maintain the order of the endpoints in doubly linked lists, this can be done in $O(1)$ time. Therefore we only spend $O(1)$ time per node of $T$.

Conversely assume that $D$ is a chord diagram of $G$ with reference chord $r$. We again process the tree $T$ in bottom-up order. As before, if $T$ contains only a single inner node $\mu$, then $c(\mu) = D$ is the desired configuration. Otherwise let $\mu$ be an inner node whose children are all leaves. We denote the set of leaves of $\mu$ by $L$. Note that $(L, V \setminus L)$ is a split of $G$. Let $T'$ be the tree obtained from $T$ by replacing $\mu$ and its leaves by a single leaf $v$. Using simple flags, we can decide for an endpoint of a chord of $D$ in constant time whether it belongs to a vertex of $L$ and if so, whether it has already been processed. We now treat the endpoints of the vertices in $L$ one by one. For the first endpoint $e$ we obtain in this way, we scan to the left and right in the doubly linked list of $D$ starting at $e$. In this way we determine a sublist $[e_1, e_2]$ of $D$ such that all endpoints that lie clockwise between $e_1, e_2$ belong to vertices of $L$ and the predecessor of $e_1$ and the successor of $e_2$ do not. We mark all endpoints that we encounter in this way as processed. We then scan further the leaves of $L$. We do find another endpoint $f$ that is not yet processed since otherwise $G$ is not connected. We similarly determine a sublist $[f_1, f_2]$ around $f$ so that all endpoints that lie clockwise between $f_1, f_2$ belong to vertices of $L$ and the predecessor of $f_1$ and the successor of $f_2$ do not. If there is an unprocessed endpoint left, this means $D$ does not respect split $(L, V \setminus L)$ and we can reject. Hence, assume no unprocessed endpoint remains.

We thus have partitioned the endpoints of the chords of $L$ into two disjoint sublists $[e_1, e_2]$ and $[f_1, f_2]$ of $D$. Then let $D'$ be the diagram obtained from $D$ by replacing the sublists $[e_1, e_2]$, $[f_1, f_2]$ each with $v$. We recursively compute a configuration $c'$ of $T'$ with $D(c') = D'$ where for each chord $u$ the end $\mathring{u}$ is stored. If this succeeds, we obtain the desired configuration $c$ as follows. For each inner node $\nu \neq \mu$ we set $c(\nu) = c'(\nu)$. Since we know from each predecessor and successor $u$ of $\mathring{v}$ and $\hat{v}$ whether it is $\mathring{u}$, we can set $c(\mu) = r[e_1, e_2]r[f_1, f_2]$ or $c(\mu) = r[f_1, f_2]r[e_1, e_2]$ according to these predecessors and successors, where $r$ is the reference chord of skel$(\mu)$. By construction it is $D(c) = D$. Note that any other choice of $c(\mu)$ or $D(c_{|T-\mu})$ results in a chord diagram different from $D$, since $[e_1, e_2]$ and $[f_1, f_2]$ are non-empty and separated by chords not in $L$. We can then first iterate through the list of $[e_1, e_2]$, $[f_1, f_2]$ that replaces $\mathring{v}$ and then other one to store each endpoint $\mathring{u}$.

The time spent to compute $T'$, $D'$ as well to modify $c'$ into $c$ is proportional to $|L|$. Therefore the algorithm runs in linear time. □

## 3.2 Configurations of Canonical Split Trees

In this section we describe the (oriented) chord diagrams for degenerate and prime connected circle graphs. This provides a compact way to describe all chord diagrams of a connected circle graph with a canonical split tree where we associate each inner node with the necessary information. We can describe a chord diagram of a connected degenerate circle graph $G = (V, E)$ with reference chord $r \in V$ with a cyclic permutation of $V$ using the following mapping $\phi_{G,r}$ to chord diagrams of $G$ with reference chord $r$. If $G$ is a clique, we set $\phi_{G,r}(r\rho) = r\rho r\rho$. If $G$ is a star with center $x$, we set $\phi_{G,r}(x\rho\sigma) = \rho^{\text{rev}}x\rho\sigma x\sigma^{\text{rev}}$ where $\rho$ ends with $r$ or is empty if $x = r$.

**Lemma 8** *For a connected degenerate circle graph $G = (V, E)$ with $r \in V$ the map $\phi_{G,r}$ is a bijection.*

***Proof*** Two chords cross in a chord diagram if and only if their ends are alternating. Therefore, $\phi_{G,r}$ actually maps to chord diagrams of $G$ with reference chord $r$. Observe that given a cyclic permutation $\pi$ of $V_G$, we can consider the corresponding permutation $\pi'$ of $V_G$ starting with $r$ or $x$ respectively and find it as a subword of $\phi_{G,r}(\pi)$. Namely, for cliques the first half of $\phi_{G,r}$ is $\pi'$ and for stars the word starting at $\mathring{x}$ and ending right before $\hat{x}$ is $\pi'$. This implies $\phi_{G,r}$ is injective. We further obtain surjectivity since each chord diagram for a clique or a star is of the corresponding form. □

For each type of connected degenerate circle graphs one of the operations turn, rev and turn ∘ rev has no effect, while the other ones have the same effect as reversing the corresponding permutation. For non-degenerate inner nodes id, turn, rev, turn ∘ rev yield pairwise distinct results.

**Lemma 9** *Let $G$ be a connected degenerate circle graph, $r \in V_G$, and let $\sigma$ be a cyclic permutation of $V_G$.*

1. *If $G$ is a clique, we have $\phi_{G,r}(\sigma) = turn(\phi_{G,r}(\sigma))$ and $\phi_{G,r}(\sigma^{\text{rev}}) = \text{rev}(\phi_{G,r}(\sigma)) = turn(\text{rev}(\phi_{G,r}(\sigma)))$.*
2. *If $G$ is a star with center $x = r$, we have $\phi_{G,r}(\sigma) = \text{rev}(\phi_{G,r}(\sigma))$ and $\phi_{G,r}(\sigma^{\text{rev}}) = turn(\phi_{G,r}(\sigma)) = turn(\text{rev}(\phi_{G,r}(\sigma)))$.*
3. *If $G$ is a star with center $x \neq r$, we have $\phi_{G,r}(\sigma) = turn(\text{rev}(\phi_{G,r}(\sigma)))$ and $\phi_{G,r}(\sigma^{\text{rev}}) = turn(\phi_{G,r}(\sigma)) = \text{rev}(\phi_{G,r}(\sigma))$.*

***Proof*** Let $\sigma = r\rho$.

1. If $G$ is a clique, we have $\phi_{G,r}(r\rho) = r\rho r\rho = \text{turn}(r\rho r\rho) = \text{turn}(\phi_{G,r}(r\rho))$ and $\phi_{G,r}(r\rho^{\text{rev}}) = r\rho^{\text{rev}}r\rho^{\text{rev}} = \text{rev}(r\rho r\rho) = \text{rev}(\phi_{G,r}(r\rho))$.
2. If $G$ is a star with center $x = r$, we have $\phi_{G,r}(r\rho) = r\rho r\rho^{\text{rev}} = \text{rev}(r\rho r\rho^{\text{rev}}) = \text{rev}(\phi_{G,r}(r\rho))$ and $\phi_{G,r}(r\rho^{\text{rev}}) = r\rho^{\text{rev}}r\rho = \text{rev}(r\rho r\rho^{\text{rev}}) = \text{rev}(\phi_{G,r}(r\rho))$.
3. If $G$ is a star with center $x \neq r$, let $\rho = \alpha x\beta$. We have $\phi_{G,r}(r\alpha x\beta) = r\beta^{\text{rev}}x\beta r\alpha x\alpha^{\text{rev}} = \text{turn}(\text{rev}(r\beta^{\text{rev}}x\beta r\alpha x\alpha^{\text{rev}})) = \text{turn}(\text{rev}(\phi_{G,r}(r\alpha x\beta)))$

and further $\phi_{G,r}(r\beta^{\text{rev}}x\alpha^{\text{rev}}) = r\alpha x\alpha^{\text{rev}}r\beta^{\text{rev}}x\beta = \text{turn}(r\beta^{\text{rev}}x\beta r\alpha x\alpha^{\text{rev}}) = \text{turn}(\phi_{G,r}(r\alpha x\beta))$.

Since turn and rev commute and are self-inverse, the remaining equations are implied. □

**Lemma 10** *Let $G$ be a connected circle graph, and let $D$ be a chord diagram of $G$ with reference chord $r \in V(G)$.*

1. *$G$ is a clique $\Leftrightarrow$ $turn(D) = D \Leftrightarrow turn(\text{rev}(D)) = \text{rev}(D)$.*
2. *$G$ is a star with center $r$ $\Leftrightarrow$ $\text{rev}(D) = D \Leftrightarrow turn(\text{rev}(D)) = turn(D)$.*
3. *$G$ is a star with center $x \neq r$ $\Leftrightarrow$ $turn(\text{rev}(D)) = D \Leftrightarrow turn(D) = \text{rev}(D)$.*
4. *$G$ is not degenerate $\Leftrightarrow$ $D$, $turn(D)$, $\text{rev}(D)$, $turn(\text{rev}(D))$ are pairwise distinct.*

***Proof*** Let $D = r\alpha r\beta$. If $\text{turn}(D) = D$, then we have $r\beta r\alpha = \text{turn}(D) = D = r\alpha r\beta$. This implies $\alpha = \beta$ and thus $D = r\alpha r\alpha$. Hence, we have that $H$ is a clique. If $\text{rev}(D) = D$, then we have $r\beta^{\text{rev}}r\alpha^{\text{rev}} = \text{rev}(D) = D = r\alpha r\beta$. This implies $\alpha = \beta^{\text{rev}}$ and thus $D = r\alpha r\alpha^{\text{rev}}$. Hence, we have that $H$ is a star with center $r$. If $\text{turn}(\text{rev}(D)) = D$, then we have $r\alpha^{\text{rev}}r\beta^{\text{rev}} = \text{turn}(\text{rev}(D)) = D = r\alpha r\beta$. This implies $\alpha = \alpha^{\text{rev}}$ and $\beta = \beta^{\text{rev}}$. Since $G$ is connected, $\alpha$ and $\beta$ must share a symbol $x$. Since this symbol occurs only once in $\alpha$ and in $\beta$, it is unique. Hence, we have that $G$ is a star with center $x \neq r$. Since $\phi_{G,r}$ is a bijection, we obtain the remaining statements from Lemma 9 and the fact that turn and rev are self-inverse. □

Recall that Bouchet [19] showed that the undirected chord diagram of a connected prime circle graph is unique up to reversal. We can additionally choose the orientation of the reference chord. As a shorthand, we set $\text{tr} = \{\text{id}, \text{turn}, \text{rev}, \text{turn}(\text{rev})\}$ and for any chord diagram $D$ we set $\text{tr}(D) = \{D, \text{turn}(D), \text{rev}(D), \text{turn}(\text{rev}(D))\}$.

**Lemma 11** *Let $G$ be a connected non-degenerate prime circle graph, $r \in V(G)$, and $D$ a chord diagram with reference chord $r$. Then $|\text{tr}(D)| = 4$ and $\text{tr}(D)$ is the set of chord diagrams of $G$ with reference chord $r$.*

***Proof*** By Lemma 10 we have $|\text{tr}(D)| = 4$. Let $D = r\alpha r\beta$ and let $\overline{D'}$ be the corresponding undirected chord diagram. Then $\text{turn}(D) = r\beta r\alpha$ is the only other chord diagram corresponding to $\overline{D}$ with reference chord $r$. Let $D' = \text{rev}\, D = r\beta^{\text{rev}}r\alpha^{\text{rev}}$ and let $\overline{D'}$ be the corresponding undirected chord diagram. We obtain that the only other chord diagram corresponding to $\overline{D'}$ with reference chord $r$ is $r\alpha^{\text{rev}}r\beta^{\text{rev}} = \text{turn}(D')$. As stated above, Bouchet showed that the undirected chord diagram of a connected prime circle graph is unique up to reversal. This means $\overline{D}$ and $\overline{D'}$ are the only undirected chord diagrams of $G$ and $D$, $\text{turn}(D)$, $\text{rev}(D)$, $\text{turn}(\text{rev}(D))$ are the only chord diagrams of $G$ with reference chord $r$. □

The following theorem summarizes the discussion. Recall that non-degenerate nodes of canonical split trees are prime.

**Theorem 12** *Let $G$ be a connected circle graph and let $T$ be the canonical split tree of $G$ with reference chord $r \in V(G)$. Let each non-degenerate node $\mu$ be equipped with a chord diagram of $\text{skel}(\mu)$ with reference chord $r_\mu$. There is a bijection between the chord diagrams of $G$ with reference chord $r$ and the choices of (i) applying an operation $\tau_\mu \in \text{tr}$ to each non-degenerate node $\mu$ and (i) choosing a cyclic permutation of $V(\text{skel}(\mu))$ for each degenerate node $\mu$.*

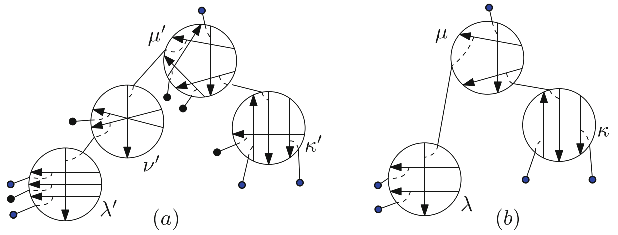

**Fig. 6** **a** configuration of a canonical split tree $T$. Leaves in $V(H)$ are colored blue. **b** split tree $T_H$ with the corresponding configuration. Node $\mu$ is degenerate in $T_H$ while $\mu'$ is prime in $T$. skel$(\kappa)$ is an isolated set while skel$(\kappa')$ is a star

## 4 Partial Representation Extension

In this section, we solve RepExt(Circle) for a circle graph $G = (V, E)$ and a chord diagram $D_H$ of an induced subgraph $H \subseteq G$ in near-linear time. The idea is the following. Assume $G$ is connected and let $T$ be the canonical split tree of $G$. We consider a split tree $T_H$ of $H$ that is obtained as a projection of $T$. Likewise, we project configurations of $T$ to configurations of $T_H$. By Theorem 6, all chord diagrams of $G$ are represented by $T$. We show that the extendable chord diagrams of $H$ are exactly the chord diagrams that correspond to projections of configurations of $T$ on $T_H$. We observe that the projection onto $H$ commutes with all relevant operations. The possible choices for $T$ described by Theorem 12 can therefore be passed to $T_H$. This yields a restricted split tree $(T_H, c'_H)$ of $H$ that describes all configurations of $H$ that can be extended to $G$. We then just check whether $(T_H, c'_H)$ represents the given chord diagram $D_H$.

Chaplick et al. [3] argue that, if $G$ is disconnected and there are distinct connected components $C, C'$ whose prerepresented chord ends alternate in $D_H$, then there is no extension of $D_H$ to $G$. Otherwise an extension exists if and only if each connected component of $G$ admits an extension. Testing this requirement as well as combining representation extensions of the different components can be done in linear time. Hence, we assume in the following that $G$ is connected. Note that $H$ may still be disconnected.

We start with a canonical split tree $T$ of $G$ rooted at a reference chord $r \in V(H)$, which by Theorem 6 represents all chord diagrams of $G$. For a chord diagram $D$ of $G$ let $D|_H$ denote the chord diagram for $H$ induced by $D$ (i.e., the chords of $H$ are placed as in $D$ with the same starting point). Let $T_H$ be the subtree of $T$ whose leaves are the vertices of $H$ and whose inner nodes are the inner nodes of $T$ that lie on a path from $r$ to some leaf in $V(H)$. For each inner node of $T_H$, we define $\text{skel}_{T_H}(\mu)$ as the subgraph of $\text{skel}_T(\mu)$ induced by the vertices $v_\mu(V(T_H))$, i.e., we keep exactly those chords that represent nodes that lead to at least one vertex of $H$ (see Fig. 6). Finally, we suppress nodes with $K_2$ as skeleton in $T_H$ by (iteratively) joining them with one of their neighbors. Note that $T_H$ is a split tree of $H$ rooted at $r$, and each inner node $\mu$ of $T_H$ stems from exactly one inner node $\mu'$ of $T$. We call $T_H$ the *projection of $T$ onto $H$*.

Let now $c$ be a configuration of $T$. We define its *projection* $c_H$ by setting $c_H(\mu) = c(\mu')|_{\mathrm{skel}_{T_H}(\mu)}$ for each node $\mu$ of $T_H$, i.e., it is the restriction of the chord diagram $c(\mu')$ to $\mathrm{skel}_{T_H}(\mu)$. Observe that the reference chord is not removed, and therefore $c_H(\mu)$ has the same reference chord as $c(\mu')$, i.e., $c_H$ is a configuration of $T_H$. The following lemma shows that the projection $(T, c) \mapsto (T_H, c_H)$ commutes with all relevant operations.

**Lemma 13** *We have (i) $D(c)|_H = D(c_H)$ and (ii) for every inner node $\mu$ of $T_H$, turn$(c_H(\mu)) =$ turn$(c(\mu'))|_{\mathrm{skel}_{T_H}(\mu)}$ and* $\mathrm{rev}(c_H(\mu)) = \mathrm{rev}(c(\mu'))|_{\mathrm{skel}_{T_H}(\mu)}$.

***Proof*** For Property (i) observe that it suffices to show that joining two diagrams commutes with the projection to a subgraph $H$. It then follows that $D(c)|_H$, where the join is projected to $H$, is the same as $D(c_H)$, where the skeletons are projected before the join, coincide.

More formally, let $H_1, H_2$ be two induced subgraphs of graphs $G_1, G_2$, respectively, and let $H = H_1 \oplus_{v_2,v_1} H_2$ and $G = G_1 \oplus_{v_2,v_1} G_2$. We show that for any chord diagrams $D_1, D_2$ of $G_1, G_2$, respectively, it is $(D_1 \oplus_{v_2,v_1} D_2)|_H = D_1|_{H_1} \oplus_{v_2,v_1} D_2|_{H_2}$. To see this, let $D_1 = \alpha v_2 \beta v_2 \gamma$ and $D_2 = v_1 \rho v_1 \sigma$ and let $\alpha', \beta', \gamma', \rho', \sigma'$ be the words obtained from $\alpha, \beta, \gamma, \rho, \sigma$ by removing all symbols for chords that are not in $V(H)$. Then we have $(D_1 \oplus_{v_2,v_1} D_2)|_H = (\alpha\rho\beta\sigma\gamma)|_H = \alpha'\rho'\beta'\sigma'\gamma'$. On the other hand, it is $D_1|_{H_1} \oplus_{v_2,v_1} D_2|_{H_2} = \alpha' v_2 \beta' v_2 \gamma' \oplus_{v_2,v_1} v_1 \rho' v_1 \sigma' = \alpha'\rho'\beta'\sigma'\gamma'$.

For Property (ii), let $H$ be an induced subgraph of $G$. Let $D = r\rho r\sigma$ be a chord diagram for $G$ with reference chord $r$ and let $\rho', \sigma'$ be the restrictions of $\rho, \sigma$ to $H$, respectively. Then $\mathrm{turn}(D)|_H = \mathrm{turn}(r\rho r\sigma)|_H = (r\sigma r\rho)|_H = r\sigma' r\rho' = \mathrm{turn}(r\rho' r\sigma') = \mathrm{turn}((r\rho r\sigma)|_H) = \mathrm{turn}(D|_H)$ and $\mathrm{rev}(D)|_H = \mathrm{rev}(r\rho r\sigma)|_H = (r\sigma^{\mathrm{rev}} r\rho^{\mathrm{rev}})|_H = r\sigma^{\mathrm{rev}} r\rho^{\mathrm{rev}}|_H = r\sigma'^{\mathrm{rev}} r\rho'^{\mathrm{rev}} = \mathrm{rev}(r\rho' r\sigma') = \mathrm{rev}(D|_H)$. □

Let $D_H$ be a chord diagram of $H$. By Theorem 6 there exists a chord diagram of $G$ that extends $D_H$ if and only if there exists a configuration $c$ of $T$ with $D(c)|_H = D_H$. By Lemma 13(i) this holds if and only if there exists a configuration $c$ of $T$ whose projection $c_H$ satisfies $D(c_H) = D(c)|_H = D_H$. We aim to find such a configuration $c$. To do this, we make use of the property from Lemma 13(ii) as follows. Let $c'$ be an arbitrary configuration of $T$. By Theorem 12, $c$ is obtained from $c'$ by (i) arbitrarily choosing a configuration for each degenerate node of $T$ and (ii) by choosing for each non-degenerate node $\mu$ one of the diagrams $c(\mu) \in \mathrm{tr}(c'(\mu))$. Note that induced subgraphs of cliques are themselves cliques and an induced subgraph of a star is either a star or an independent set. In the latter case an induced chord diagram has the form $\rho^{\mathrm{rev}}\rho\sigma\sigma^{\mathrm{rev}}$ where the center of the original star has one end between $\rho^{\mathrm{rev}}$ and $\rho$ and one end between $\sigma$ and $\sigma^{\mathrm{rev}}$. By Lemma 13(ii) it follows that $c_H$ is obtained from $c'_H$ by (i) arbitrarily choosing a configuration for each node of $T_H$ that stems from a degenerate node of $T$ and is connected, (ii) choosing a configuration of the form $\rho^{\mathrm{rev}}\rho\sigma\sigma^{\mathrm{rev}}$ for each node of $T_H$ that stems from a degenerate node of $T$ and is an independent set and (iii) by choosing for each node $\mu$ of $T_H$ that stems from a non-degenerate node in $T$ one of the diagrams $c_H(\mu) \in \mathrm{tr}(c'_H(\mu))$.

We condense these rules as follows. We label the nodes of $T_H$ as *degenerate* if they stem from a degenerate node in $T$ and as *prime* if they stem from a non-degenerate node of $T$. We call two configuration $c_H, c'_H$ of $T_H$ *equivalent* if for each prime-labeled node $\mu$ we have $c_H(\mu) \in \mathrm{tr}(c'_H(\mu))$ and for each degenerate-labeled node $\nu$ where $\mathrm{skel}(\nu)$ is an independent set $c_H(\nu)$ is of the form $\rho^{\mathrm{rev}}\rho\sigma\sigma^{\mathrm{rev}}$. We call $(T_H, c'_H)$ *the restricted split tree of* $H$ *with respect to* $G$ and say that $(T_H, c'_H)$ *represents* a diagram $D_H$ of $H$ if $D_H = D(c_H)$ for some configuration $c_H$ that is equivalent to $c'_H$. By the above observations, $D_H$ can be extended to a diagram of $G$ if and only if $D_H$ is represented by $(T_H, c'_H)$, where $c'_H$ is the projection of a configuration of $T$.

If $H$ is connected, we can compute in linear time the unique configuration $c_H$ of $T_H$ with $D(c_H) = D_H$ using Theorem 7 and then check whether it is equivalent to $c'_H$. If it is not equivalent to $c'_H$, then no such configuration exists since $c_H$ is unique. In the next section we show how to find a configuration $c_H$ represented by $(T_H, c'_H)$ with $D(C_H) = D_H$ in the case that $H$ is not connected.

### 4.1 Representation Test in Restricted Split Trees

Let $G$ be a connected circle graph with canonical split tree $T$ and a configuration $c'$ and let $(T_H, c'_H)$ be the restricted split tree of a subgraph $H$ in $G$. Given a chord diagram $D_H$ for $H$, we aim to find a configuration $c_H$ of $T_H$ with $D(c_H) = D_H$ that is equivalent to $c'_H$. To this end, we proceed similarly as in the proof of Theorem 7. However, $H$ might be disconnected. We can therefore have that when processing a node $\mu$, the two ends of the chord $v$ representing $\mu$ in its parent $\kappa$ are contiguous. For example, this happens if $\kappa$ is a star node in $T$ that lost its center in $T_H$ and $v_\kappa(\mu)$ is the leftmost leaf. If, additionally, $\mu$ is a node where all chords are at the same side of the reference chord, configuration $c_H$ is not unique. Further, we only find a single sequence $[e_1, e_2]$ of leaves of $\mu$. This also occurs, if all chords are on the same side of the reference chord in $\mu$. Without the second sequence $[f_1, f_2]$, it is too expensive to find the correct place in the new chord diagram $D'$ for the second end of the leaf chord $v$ that replaces $\mu$. For that reason, we instead work with *relaxed* chord diagrams, where not necessarily all ends are placed. More precisely, a relaxed chord diagram $D$ of a graph $G$ with reference chord $r$ is a word $D = r\alpha$ over $V(G)$ where each $v \in V(G)$ appears exactly once or twice in $r\alpha$ and for any two chords $x, y$ with two ends in $D$ we have that $x$ and $y$ alternate in $D$ if and only if $xy \in E(G)$. We say that a chord diagram $D'$ *realizes* $D$ if we can obtain $D$ from $D'$ by removing chord ends. For nodes in a restricted split tree it is easy to find configurations that realize a relaxed chord diagram.

**Lemma 14** *Let $\mu$ be an inner node of a restricted split tree $(T_H, c'_H)$ and let $D_H$ be a relaxed chord diagram of* $\mathrm{skel}(\mu)$ *with the same reference chord $r_\mu$ as $c'_H(\mu)$. Then it can be tested in $O(|V(\mathrm{skel}(\mu))|)$ time whether there exists a configuration $c_H$ of $T_H$ with $c_H(\mu) = D_H$ that is equivalent to $c'_H$. If it exists, $c_H(\mu)$ can be computed in the same time.*

***Proof*** If $\mu$ is a prime-labeled node, we do for each $D'_H \in \mathrm{tr}(c'_H(\mu))$ the following. We traverse $D'_H$ and $D_H$ starting at the starting point. Whenever we do not encounter an end of a chord $v$ in $D_H$ that we do encounter in $D'_H$, we insert that end in $D_H$. If both ends of $v$ are already contained in $D_H$, $D'_H$ does not realize $D_H$. If we finish the traversal, $D'_H$ realizes $D_H$ by definition and we can choose $c_H(\mu) = D'_H$. If no

$D'_H \in \mathrm{tr}(c'_H(\mu))$ realizes $D_H$, then reject. Note that the placements of all added chord ends are necessary and thus this approach is correct.

If $\mu$ is a degenerate-labeled node that stems from a clique, then $\mathrm{skel}(\mu)$ is itself a clique. If $D_H$ contains no chord end twice, we just set $c_H(\mu) = D_H D_H$. Otherwise, we iteratively choose a maximal sequence $[e_1, e_2]$ of chord ends where all chords have only one end (the one in $[e_1, e_2]$). First consider the case where there exists a predecessor $f$ of $[e_1, e_2]$, then since $[e_1, e_2]$ is maximal, chord $f$ has two ends in $D_H$. Insert a copy of $[e_1, e_2]$ after the other end of $f$. Thereby all chords in $[e_1, e_2]$ intersect $f$ and each other, and further all other chords with two ends also intersect the chords in $[e_1, e_2]$ since they intersect $f$. If $[e_1, e_2]$ has no predecessor, then it has a successor $g$ since there is a chord with two ends. In that case we can argue similarly that a copy of $[e_1, e_2]$ can be inserted in front of the other end of $g$. Hence, in the end we obtain a chord diagram $c_H(\mu)$ of clique $\mathrm{skel}(\mu)$ that realizes $D_H$.

If $\mu$ is a degenerate-labeled node and $\mathrm{skel}(\mu)$ is a star with center $x$, we start at an end of $x$ in $D_H$ and traverse simultaneously in both directions (eventually going from the end of the word to the start or the other way around, respectively) until all ends are traversed (except possibly the second end of $x$). At each end of a chord $v$ with no second end, insert that second end where the other traversal is at that moment. When a chord with two ends is reached, wait in front of that chord until the other traversal also reaches that chord and then skip that chord in both traversals. If the traversals wait at different chords we reject. In that case $D_H$ induces the subword $xaabb$ or $xabab$ (or some cyclic shifted version of these words) which cannot be realized by a star with center $x$. If $x$ has a second end, the traversals meet and end there. Otherwise, add the second end of $x$ where the traversals meet. Thereby, all chords for leaves intersect $x$ and no other intersection occurs. By construction we obtain a chord diagram $c_H(\mu)$ that realizes $D_H$.

If $\mu$ is a degenerate-labeled node and $\mathrm{skel}(\mu)$ is an independent set, then we choose a place that – ignoring ends of chords with only one end in $D_H$ – is not neighboring two different chords. We then place there a chord end of an artificial center and proceed as for stars above. Note that if both traversals wait at different chords, then $D_H$ induces the subword $aabbcc$ or $aabcbc$ (or some cyclic shifted version of that word) with the chord bordering $x$ and the two chords where the traversals wait as $a$, $b$ and $c$. Hence, $D_H$ can in that case not be extended to the form $\rho^{\mathrm{rev}}\rho\sigma\sigma^{\mathrm{rev}}$. If the process finishes, we remove the artificial center and obtain $c_H(\mu)$ in the desired form.

Since we essentially only do traversals, the running time is clearly in $O(|V(\mathrm{skel}(\mu))|)$. □

**Lemma 15** *Let $(T_H, c'_H)$ be a restricted split tree of a graph $H$ with respect to a connected graph $G$ and let $D_H$ be a chord diagram of $H$ with the root $r$ of $T_H$ as reference chord. Then it can be tested in linear time whether there exists a configuration $c_H$ of $T_H$ with $D(c_H) = D_H$. If it exists, such a configuration $c_H$ can also be computed in linear time.*

***Proof*** We actually prove a stronger statement where $D_H$ can be a relaxed chord diagram with reference chord $r$ and $D(c_H)$ realizes $D_H$. We proceed similar as in the proof of Theorem 7.

We process the tree $T_H$ in bottom-up order. If $T_H$ contains only a single inner node $\mu$, then Lemma 14 provides the result. Otherwise let $\mu$ be an inner node whose children are all leaves. We denote the set of leaves of $\mu$ by $L$. Note that $(L, V \setminus L)$ is a split of $H$. Let $(T'_H, c''_H)$ be the restricted split tree obtained from $(T_H, c'_H)$ by replacing $\mu$ and its leaves by a single leaf $v$ with $c''_H(\nu) = c'_H(\nu)$ for every node $\nu$ of $T'_H$. Using simple flags, we can decide for an endpoint of a chord of $D_H$ in constant time whether it belongs to a vertex of $L$ and if so, whether it has already been processed. We now treat the endpoints of the vertices in $L$ one by one. For the first endpoint $e$ we obtain in this way, we scan to the left and right in the doubly linked list of $D_H$ starting at $e$. In this way we determine a sublist $[e_1, e_2]$ of $D_H$ such that all endpoints that lie clockwise between $e_1, e_2$ belong to vertices of $L$ and the predecessor of $e_1$ and the successor of $e_2$ do not. We mark all endpoints that we encounter in this way as processed. We then scan further the leaves of $L$. If we find another endpoint $f$ that is not yet processed, we similarly determine a sublist $[f_1, f_2]$ around $f$ so that all endpoints that lie clockwise between $f_1, f_2$ belong to vertices of $L$ and the predecessor of $f_1$ and the successor of $f_2$ do not. If there is an unprocessed endpoint left, this means $D_H$ does not respect split $(L, V \setminus L)$ and we can reject. Hence, assume no unprocessed endpoint remains.

If we found $f$, we thus have partitioned the endpoints of the chords of $L$ into two disjoint sublists $[e_1, e_2]$ and $[f_1, f_2]$ of $D_H$. Then let $D'$ be the relaxed chord diagram obtained from $D_H$ by replacing the sublists $[e_1, e_2]$, $[f_1, f_2]$ each with $v$.

Note that $D(c_{H|T-\mu})$ must realize $D'$, since $[e_1, e_2]$ and $[f_1, f_2]$ are non-empty and separated by chords not in $L$ in $D_H$. We recursively compute a configuration $c'$ of $T'_H$ equivalent to $c''_H$ such that $D(c')$ realizes $D'$ and for each chord $u$ the end $\mathring{u}$ is stored. If this succeeds, we obtain the desired configuration $c_H$ as follows. For each inner node $\nu \neq \mu$ we set $c_H(\nu) = c'(\nu)$. Since we know from each predecessor and successor $u$ of $\mathring{v}$, $\hat{v}$ whether it is $\mathring{u}$, we can define a relaxed chord diagram $D'_\mu = r[e_1, e_2]r[f_1, f_2]$ or $D'_\mu = r[f_1, f_2]r[e_1, e_2]$ depending on whether $\mathring{v}$ is at the place of $[e_1, e_2]$ or $[f_1, f_2]$. Note that $c_H(\mu)$ must realize $D'_\mu$ since at the places of $\mathring{v}$ and $\hat{v}$ correpsonding words must be inserted. Hence, we obtain $c_H(\mu)$ from $D'_\mu$ by Lemma 14. By construction $D(c_H)$ realizes $D_H$.

In the case where $f$ was not found, we have that $[e_1, e_2]$ contains all chord ends of $L$. Since this might only determine one end of $v$ in $D'$, we have to proceed more carefully. Namely, we first check whether it suffices to fix one end of $v$ in $D'$ by checking whether $\mu$ admits a diagram where the endpoints are in different spots, or whether they have to be in the same position. Depending on that outcome we then replace $[e_1, e_2]$ by one or two ends of $v$ for the recursive call. Finally, we adapt $c_H(\mu)$. In detail we do this as follows. We first check whether we may choose a chord diagram $D_\mu$ that realizes $D'_\mu = r[e_1, e_2]r$ with Lemma 14. In that case, we may instead also choose $\mathrm{turn}(D_\mu)$, which realizes $rr[e_1, e_2]$. We then define $D'$ as the relaxed diagram obtained from $D_H$ by replacing $[e_1, e_2]$ with a single end of $v$ and not fixing the second end of $v$. Note that $D(c_{H|T_H-\mu})$ must realize $D'$, since $[e_1, e_2]$ is non-empty and to have these chord ends in the corresponding position, at least one end of $v$ has to be placed there in $D(c_{H|T_H-\mu})$. Again, we recursively compute a configuration $c'$ of $T'_H$ equivalent to $c''_H$ such that $D(c')$ realizes $D'$ where for each chord $u$ the end $\mathring{u}$ is stored. As argued above, such a configuration $c'$ must exist, or we can reject. If this succeeds, we obtain the desired configuration $c$ as follows. For each inner node $\nu \neq \mu$

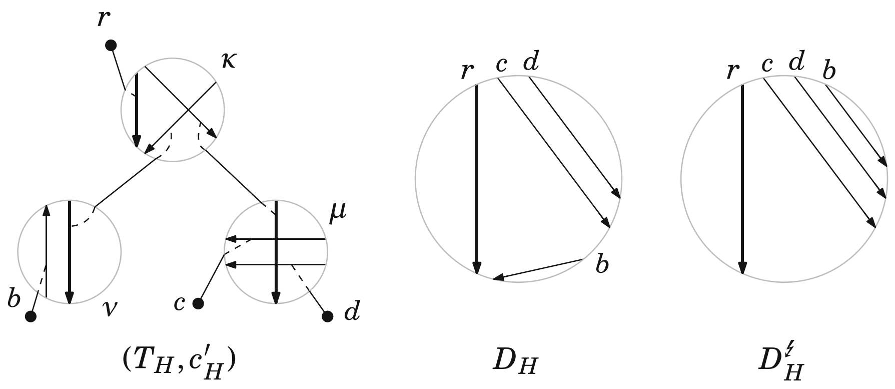

**Fig. 7** Example where replacing $[e_1, e_2]$ by a single $v$ may result in providing a false configuration $c_H$. We have $[e_1, e_2] = cddc$. Setting $c_H = c'_H$ provides $D_H$. If we replace $cddc$ by a single $v$ in $D'$, then $\nu$ can be reversed resulting in chord diagram to the right (turning $\mu$ does not help) while $D(c_H|_{\{\kappa,\nu\}})$ realizes $D'$ and $c_H(\mu) = rcdrdc$

we set $c_H(\nu) = c'(\nu)$. We further set $c_H(\mu) = D_\mu$ if $[e_1, e_2]$ was replaced by $\mathring{v}$ and we set $c_H(\mu) = \text{turn}(D_\mu)$ if $[e_1, e_2]$ was replaced by $\hat{v}$. By construction, $D(c_H)$ realizes $D_H$. □

In the case that we may not choose a chord diagram for $\mu$ that realizes $D'_\mu = r[e_1, e_2]r$ (or $rr[e_1, e_2]$), the only way to obtain $[e_1, e_2]$ as a sequence in $D(c_H)$ is if both ends of $v$ are contiguous in $D(c_{H|T_H - \mu})$ (where they then have to be replaced by $[e_1, e_2]$). Thus, define $D'$ as the relaxed diagram obtained from $D_H$ by replacing $[e_1, e_2]$ with both ends of $v$. Note that replacing $[e_1, e_2]$ by just one end of $v$ would allow the other end to be non-contiguous in $D(c_{H|T_H - \mu})$; see Fig. 7 for an example.

This further means that any diagram $c_H(\mu)$ that realizes $r[e_1, e_2]$ provides $D_H$ when joined with $D(c_{H|T_H - \mu})$. Hence, we recursively compute a configuration $c'$ of $T'_H$ equivalent to $c''_H$ such that $D(c')$ realizes $D'$ where for each chord $u$ the end $\mathring{u}$ is stored. We further use Lemma 14 to find an allowed chord diagram $D_\mu$ for $\mu$ that realizes $D'_\mu$. If this succeeds, we obtain the desired configuration $c$ by setting $c(\nu) = c'(\nu)$ for each inner node $\nu \neq \mu$ and $c_H(\mu) = D_\mu$. By construction, $D(c_H)$ realizes $D_H$.

We can either way iterate through $c_H(\mu)$ to store each endpoint $\mathring{u}$. The time spent to compute $(T'_H, D', D'_\mu)$ as well as to modify $c'$ into $c_H$ is proportional to $|L|$. Therefore the algorithm runs in linear time.

## 4.2 The Actual Extension

We can conclude with the following theorem using a straightforward extension.

**Theorem 16** *Given $T = ST(G)$ and a chord diagram $D$ of $G$,* RepExt(Circle) *can be solved in linear time. In the positive case a representation of $G$ that extends the given diagram $D_H$ of $H$ can be computed in the same running time.*

***Proof*** From $D$ we compute in linear time a configuration $c'$ of $T$ with $D(c') = D$ using Theorem 7. From $T$ and $c'$, we compute in linear time the projection to the restricted split-tree $(T_H, c'_H)$ of $H$. With Lemma 15, we test whether it represents the given diagram $D_H$ of $H$ and obtain a configuration $c_H$ equivalent to $c'_H$ with $D(c_H) = D_H$. We now define a configuration $c$ of $T$ as follows. For each prime-labeled node $\mu$ of $T_H$, we define $c(\mu') = \tau(c'(\mu'))$ where $\tau \in$ tr such that $c_H(\mu) = \tau(c'_H(\mu))$. For each degenerate-labeled node $\mu$ of $T_H$, we choose a configuration as follows. If $\text{skel}_H(\mu)$ is connected, we have by Lemma 8 that $c_H(\mu) = \phi_{\text{skel}_H(\mu),r}(\sigma)$ for some cyclic permutation $\sigma$ of $V(\text{skel}_{T_H}(\mu))$. We create a permutation $\sigma'$ of $V(\text{skel}_T(\mu))$ by appending the elements of $V(\text{skel}_T(\mu'))\setminus V(\text{skel}_{T_H}(\mu))$ to $\sigma$ in an arbitrary order. We then set $c(\mu') = \phi_{H,r}(\sigma')$. If $\text{skel}_{T_H}(\mu)$ is not connected, then $\text{skel}_T(\mu')$ is a star where the reference chord $r_{\mu'}$ is not the center $x$ and $\text{skel}_{T_H}(\mu)$ does not contain $x$. In that case $c_H(\mu)$ is of the form $c_H(\mu) = \rho^{\text{rev}}\rho\sigma\sigma^{\text{rev}}$. Then set $c(\mu') = \rho^{\text{rev}}x\rho\sigma\alpha x\alpha^{\text{rev}}$, where $\alpha$ are the elements of $V(\text{skel}_T(\mu)) \setminus V(\text{skel}_{T_H}(\mu))$. Finally, for each node $\mu'$ of $T$ that is not contained in $T_H$, we set $c(\mu') = c'(\mu')$. By construction, we have that $c_H$ is the projection of $c$, and therefore $D(c)|_H = D(c_H) = D_H$, i.e., $D(c)$ is the desired representation of $G$. Clearly the amount of work per skeleton is linear, and therefore the overall running time is linear. □

If the canonical split tree of $G$ or a chord diagram of $G$ are not yet available, we compute them in $O((n+m)\alpha(n+m))$ time using the algorithm of Gioan et al. [1, 15].

## 5 Conclusion

We have developed a data structure that compactly represents all chord diagrams for a connected circle graph. As an application, we have shown how to solve the partial representation extension problem for circle graphs in almost linear time, improving over the $O(n^3)$ algorithm of Chaplick et al. [3]. Using a reduction of Chaplick et al. this also solves the extension problem for permutation graphs in near linear time, improving over the $O(n^3)$ algorithms of Chaplick et al. [3] and Klavík et al. [11]. By now Münch et al. provided a linear-time algorithm [20]. Our data structure may also be useful when seeking restricted chord diagrams that satisfy additional constraints. For example, we believe that it is possible to significantly simplify the circular-arc graph recognition of Hsu et al. [21].

**Author Contributions** All authors wrote the main manuscript text and prepared figures. All authors reviewed the manuscript.

**Funding** Open access publishing supported by the National Technical Library in Prague. This work was funded by Grant RU 1903/3-1 of the German Research Foundation (DFG). The authors have no competing interests to declare that are relevant to the content of this article.

**Data availibility** Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

## Declarations

**Competing interests** The authors declare no competing interests.

## References

1. Gioan, E., Paul, C., Tedder, M., Corneil, D.: Practical and efficient circle graph recognition. Algorithmica **69**(4), 759–788 (2014). https://doi.org/10.1007/s00453-013-9745-8
2. Spinrad, J.: Recognition of circle graphs. J. Algorithms **16**(2), 264–282 (1994)
3. Chaplick, S., Fulek, R., Klavík, P.: Extending partial representations of circle graphs. J. Graph Theory **91**(4), 365–394 (2019)
4. Klavík, P., Kratochvíl, J., Vyskočil, T.: Extending partial representations of interval graphs. In: Ogihara, M., Tarui, J. (eds.) Theory and Applications of Models of Computation: 8th Annual Conference, TAMC 2011, Tokyo. Proceedings, pp. 276–285. Springer, Cham. (2011). https://doi.org/10.1007/978-3-642-20877-5_28
5. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T., Vyskočil, T.: Extending partial representations of interval graphs. Algorithmica **78**(3), 945–967 (2017)
6. Angelini, P., Di Battista, G., Frati, F., Jelínek, V., Kratochvíl, J., Patrignani, M., Rutter, I.: Testing planarity of partially embedded graphs. ACM Trans. Algorithms **11**(4), 32–13242 (2015). https://doi.org/10.1145/2629341
7. Patrignani, M.: On extending a partial straight-line drawing. Int. J. Found. Comput. Sci. **17**(5), 1061–1070 (2006). https://doi.org/10.1142/S0129054106004261
8. Arroyo, A., Derka, M., Parada, I.: Extending simple drawings. In: Archambault, D., Tóth, C.D. (eds.) Proceedings of the 27th International Symposium on Graph Drawing and Network Visualization (GD'19). Lecture Notes in Computer Science, vol. 11904, pp. 230–243. Springer, Cham. (2019). https://doi.org/10.1007/978-3-030-35802-0_18
9. Eiben, E., Ganian, R., Hamm, T., Klute, F., Nöllenburg, M.: Extending partial 1-planar drawings. In: Czumaj, A., Dawar, A., Merelli, E. (eds.) Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP'20). LIPIcs, vol. 168, pp. 43–14319. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2020). https://doi.org/10.4230/LIPIcs.ICALP.2020.43
10. Klavík, P., Kratochvíl, J., Otachi, Y., Rutter, I., Saitoh, T., Saumell, M., Vyskočil, T.: Extending partial representations of proper and unit interval graphs. In: Algorithm Theory–SWAT 2014, pp. 253–264. Springer, Cham. (2014)
11. Klavík, P., Kratochvíl, J., Krawczyk, T., Walczak, B.: Extending partial representations of function graphs and permutation graphs. In: Epstein, L., Ferragina, P. (eds.) Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12). Lecture Notes in Computer Science, vol. 7501, pp. 671–682. Springer, Cham. (2012). https://doi.org/10.1007/978-3-642-33090-2_58
12. Krawczyk, T., Walczak, B.: Extending partial representations of trapezoid graphs. In: Bodlaender, H.L., Woeginger, G.J. (eds.) Proceedings of the 43rd International Workshop on Graph-Theoretic Concepts in Computer Science. Lecture Notes in Computer Science, vol. 10520, pp. 358–371. Springer, Cham. (2017). https://doi.org/10.1007/978-3-319-68705-6_27

13. Battista, G.D., Tamassia, R.: On-line graph algorithms with SPQR-trees. In: Paterson, M.S. (ed.) Proceedings of the 17th International Colloquium on Automata, Languages, and Programming. Lecture Notes in Computer Science, vol. 443, pp. 598–611. Springer, Cham. (1990)
14. Gallai, T.: Transitiv orientierbare Graphen. Acta Mathematica Academiae Scientiarum Hungarica **18**, 25–66 (1967)
15. Gioan, E., Paul, C., Tedder, M., Corneil, D.: Practical and efficient split decomposition via graph-labelled trees. Algorithmica **69**(4), 789–843 (2014). https://doi.org/10.1007/s00453-013-9752-9
16. Kalisz, V., Klavík, P., Zeman, P.: Circle graph isomorphism in almost linear time. In: Du, D., Du, D., Wu, C., Xu, D. (eds.) Theory and Applications of Models of Computation - 17th Annual Conference, TAMC 2022. Lecture Notes in Computer Science, vol. 13571, pp. 176–188. Springer, Cham. (2022). https://doi.org/10.1007/978-3-031-20350-3_15
17. Courcelle, B.: Circle graphs and monadic second-order logic. J. Appl. Log. **6**(3), 416–442 (2008). https://doi.org/10.1016/j.jal.2007.05.001
18. Cunningham, W.H.: Decomposition of directed graphs. SIAM J. Algebraic Discrete Methods **3**(2), 214–228 (1982). https://doi.org/10.1137/0603021
19. Bouchet, A.: Reducing prime graphs and recognizing circle graphs. Combinatorica **7**(3), 243–254 (1987)
20. Münch, M., Rutter, I., Stumpf, P.: Partial and simultaneous transitive orientations via modular decompositions. In: Bae, S.W., Park, H. (eds.) 33rd International Symposium on Algorithms and Computation, ISAAC. LIPIcs, vol. 248, pp. 51–15116. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2022). https://doi.org/10.4230/LIPIcs.ISAAC.2022.51
21. Hsu, W.L.: $O(m \cdot n)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. SIAM J. Comput. **24**(3), 411–439 (1995). https://doi.org/10.1137/S0097539793260726