Algorithmica (2024) 86:1263–1292
https://doi.org/10.1007/s00453-023-01188-y

Partial and Simultaneous Transitive Orientations via Modular Decompositions

Miriam Münch[1] · **Ignaz Rutter**[1] · **Peter Stumpf**[1]

Received: 7 November 2022 / Accepted: 6 November 2023 / Published online: 14 December 2023
© The Author(s) 2023

Abstract
A natural generalization of the recognition problem for a geometric graph class is the problem of extending a representation of a subgraph to a representation of the whole graph. A related problem is to find representations for multiple input graphs that coincide on subgraphs shared by the input graphs. A common restriction is the sunflower case where the shared graph is the same for each pair of input graphs. These problems translate to the setting of comparability graphs where the representations correspond to transitive orientations of their edges. We use modular decompositions to improve the runtime for the orientation extension problem and the sunflower orientation problem to linear time. We apply these results to improve the runtime for the partial representation problem and the sunflower case of the simultaneous representation problem for permutation graphs to linear time. We also give the first efficient algorithms for these problems on circular permutation graphs.

Keywords Representation extension · Simultaneous representation · Comparability graph · Permutation graph · Circular permutation graph · Modular decomposition

1 Introduction

Representations and drawings of graphs have been considered since graphs have been studied [1]. A *geometric intersection representation* of a graph $G = (V, E)$ with regards to a class of geometric objects $\mathcal{C}$, is a map $R\colon V \to \mathcal{C}$ that assigns objects of $\mathcal{C}$ to the vertices of G such that G contains an edge uv if and only if the intersection $R(u) \cap$

✉ Miriam Münch
muenchm@fim.uni-passau.de

Ignaz Rutter
rutter@fim.uni-passau.de

Peter Stumpf
stumpf@fim.uni-passau.de

[1] Faculty of Computer Science and Mathematics, University of Passau, Passau, Germany

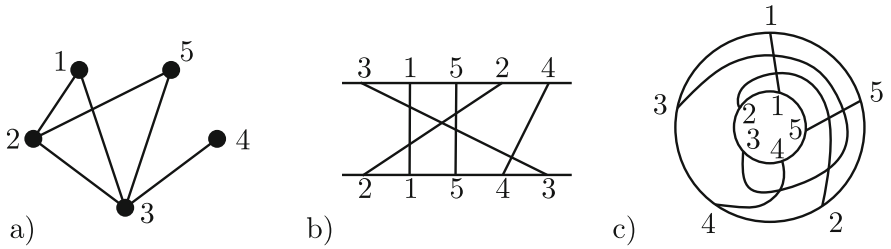

Fig. 1 **a** A graph G with **b** a permutation diagram and **c** a circular permutation diagram

$R(v)$ is non-empty. In this way, the class $\mathcal{C}$ gives rise to a class of graphs, namely the graphs that admit such a representation. As an example, consider *permutation diagrams* where $\mathcal{C}$ consists of segments connecting two parallel lines ℓ_1, ℓ_2, see Fig. 1b, which defines the class PERM of *permutation graphs*. Similarly, the class CPERM of *circular permutation graphs* is obtained by replacing ℓ_1, ℓ_2 with concentric circles and the geometric objects with curves from ℓ_1 to ℓ_2 that pairwise intersect at most once; see Fig. 1c.

A key problem in this context is the *recognition problem*, which asks whether a given graph admits such a representation for a fixed class $\mathcal{C}$. Klavík et al. introduced the *partial representation extension problem* (REPEXT($\mathcal{C}$)) for intersection graphs where a representation R' is given for a subset of vertices $W \subseteq V$ and the question is whether R' can be extended to a representation R of G, in the sense that $R|_W = R'$ [1]. They showed that REPEXT can be solved in linear time for interval graphs. The problem has further been studied for proper/unit interval graphs [2], function and permutation graphs (PERM) [3], circle graphs [4], chordal graphs [5], and trapezoid graphs [6]. Related extension problems have also been considered, e.g., for planar topological [7, 8] and straight-line [9] drawings, for 1-planar drawings [10], for contact representations [11], and for rectangular duals [12].

A related problem is the *simultaneous representation problem* (SIMREP($\mathcal{C}$)) where input graphs $G_1 \ldots, G_r$ that may share subgraphs are given and the question is whether they have representations $R_1, \ldots, R_r$ such that for $i, j \in \{1, \ldots, r\}$ the shared graph $H = G_i \cap G_j$ has the same representation in R_i and R_j, i.e., $R_i|_{V(H)} = R_j|_{V(H)}$. If more than two input graphs are allowed, usually the *sunflower case* (SIMREP*($\mathcal{C}$)) is considered, where the shared graph $H = G_i \cap G_j$ is the same for any $i \neq j \in \{1, \ldots, r\}$. I.e., here the question is whether H has a representation that can be simultaneously extended to $G_1, \ldots, G_r$. Simultaneous representations were first studied in the context of planar drawings [13, 14], where the goal is to embed each input graph without edge crossings while shared subgraphs have the same induced embedding. Unsurprisingly, many variants are NP-complete [15–18].

Motivated by applications in visualization of temporal relationships, and for overlapping social networks or schedules, DNA fragments of similar organisms and adjacent layers on a computer chip, Jampani and Lubiw introduced the problem SIMREP for intersection graphs [19]. They provided polynomial-time algorithms for two chordal graphs and for SIMREP*(PERM). They also showed that in general SIMREP is NP-complete for three or more chordal graphs. The problem was also studied for

Table 1 Known runtimes on the left and new runtimes on the right. For SIMREP* we set $n = \sum_{i=1}^{r} |V(G_i)|$ and $m = \sum_{i=1}^{r} |E(G_i)|$

	REPEXT	SIMREP*
COMP	$O((n+m)\Delta)$ [3]	$O(nm)$ [19]
PERM	$O(n^3)$ [3]	$O(n^3)$ [19]
CPERM	open	open

	REPEXT	SIMREP*
COMP	$O(n+m)$	$O(n+m)$
PERM	$O(n+m)$	$O(n+m)$
CPERM	$O(n+m)$	$O(n^2)$

We use REPEXT(COMP) and SIMREP*(COMP) to refer to ORIENTEXT and SIMORIENT*, respectively, in a slight abuse of notation

interval graphs [20–22], proper/unit interval graphs [23], circular-arc graphs [22] and circle graphs [4].

Many of the considered graph classes are related to the class COMP of *comparability graphs* [24]. An *orientation* O of a graph $G = (V, E)$ assigns to each edge of G a direction. The orientation O is *transitive* if $uv, vw \in O$ implies $uw \in O$. A comparability graph is a graph for which there is a *transitive orientation*. A *partial orientation* is an orientation of a (not necessariliy induced) subgraph of G. Similar to REPEXT, SIMREP*and SIMREP, the problems ORIENTEXT, SIMORIENT* and SIMORIENT for comparability graphs ask for a transitive orientation of a graph that extends a given partial orientation and for transitive orientations that coincide on the shared graph, respectively. The key ingredient for the $O(n^3)$ algorithm solving REPEXT(PERM) by Klavík et al. [3] is a polynomial-time solution for ORIENTEXT based on the transitive orientation algorithm by Gilmore and Hoffman [25]. Likewise, the $O(n^3)$ algorithm solving SIMREP*(PERM) by Jampani and Lubiw [19] is based on a polynomial-time algorithm for SIMORIENT* based on the transitive orientation algorithm by Golumbic [24].

Contribution and Outline. In Sect. 2, we introduce modular decompositions which can be used to describe certain subsets of the set of all transitive orientations of a graph, e.g., those that extend a given partial representation. Based on this, we give a simple linear-time algorithm for ORIENTEXT in Sect. 3. Afterwards, in Sect. 4, we develop an algorithm for intersecting subsets of transitive orientations represented by modular decompositions and use this to give a linear-time algorithm for SIMORIENT*. In Sect. 5 we give linear-time algorithms for REPEXT(PERM) and SIMREP*(PERM), improving over the $O(n^3)$-algorithms of Klavík et al. and Jampani and Lubiw, respectively. We also give the first efficient algorithms for REPEXT(CPERM) and SIMREP*(CPERM) in Sect. 6. Table 1 gives an overview of the state of the art and our results. In Sect. 7 we show that the simultaneous orientation problem and the simultaneous representation problem for permutation graphs are both NP-complete in the non-sunflower case.

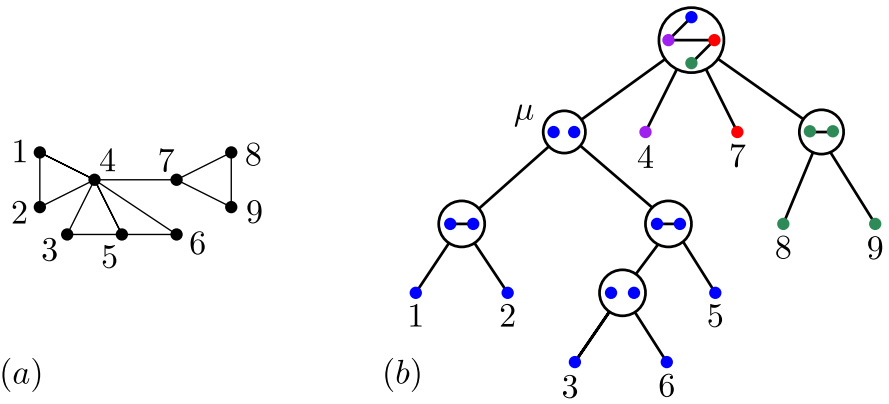

Fig. 2 **a** A graph G. **b** The canonical modular decomposition of G with $L(\mu) = \{1, 2, 3, 5, 6\}$

2 Modular Decompositions

Let $G = (V, E)$ be an undirected graph. We write $G[U]$ for the subgraph induced by a vertex set $U \subseteq V$. For a rooted tree T and a node μ of T, we write $T[\mu]$ for the subtree of T with root μ and $L(\mu)$ for the leaf-set of $T[\mu]$.

A *module* of G is a non-empty set of vertices $M \subseteq V$ such that every vertex $u \in V \setminus M$ is either adjacent to all vertices in M or to none of them. The singleton subsets and V itself are called the *trivial modules*. A module $M \subsetneq V$ is *maximal*, if there exists no module M' such that $M \subsetneq M' \subsetneq V$. If G has at least three vertices and no non-trivial modules, then it is called *prime*. We call a rooted tree T with root ρ and $L(\rho) = V$ a *(general) modular decomposition* for G if for every node μ of T the set $L(\mu)$ is a module; see Fig. 2.

Observe that for any two nodes $\mu_1, \mu_2 \in T$ such that neither of them is an ancestor of the other, G contains either all edges with one endpoint in $L(\mu_1)$ and one endpoint in $L(\mu_2)$ or none of them. For two vertices $u, v \in V$ we denote the lowest common ancestor of their corresponding leaves in T by $\mathrm{lca}_T(u, v)$. For a set of leaves L, we denote the lowest common ancestor by $\mathrm{lca}_T(L)$.

With each inner node μ of T we associate a *quotient graph* $G[\mu]$ that is obtained from $G[L(\mu)]$ by contracting $L(\nu)$ into a single vertex for each child ν of μ; see Fig. 2. In the rest of this paper we identify the vertices of $G[\mu]$ with the corresponding children of μ. Every edge $uv \in E$ is *represented* by exactly one edge $\mathrm{rep}_T(uv)$ in one of the quotient graphs of T, namely in the quotient graph $G[\mu]$ of the lowest common ancestor μ of u and v. More precisely, if ν and λ are the children of μ with $u \in L(\nu)$ and $v \in L(\lambda)$, then $\mathrm{rep}_T(uv) = \nu\lambda$. For an oriented edge uv, $\mathrm{rep}_T(uv)$ is also oriented towards its endpoint ν with $v \in L(\nu)$. If T is clear from the context, the subscript can be omitted. Let μ be a node in T. For a vertex $u \in L(\mu)$ we denote the child λ of μ with $u \in L(\lambda)$ by $\mathrm{rep}_\mu(u)$.

A node μ in a modular decomposition T is called *empty*, *complete* or *prime* if the quotient graph $G[\mu]$ is *empty*, *complete* or *prime*, respectively. By $K(T)$, $P(T)$ we denote the set of all complete and prime nodes in T, respectively. For every graph G

there exists a uniquely defined modular decomposition, that we call *the canonical modular decomposition* of G, introduced by Gallai [26], such that each quotient graph is either prime, complete or empty and, additionally, no two adjacent nodes are both complete or are both empty; see Fig. 2. Note that in the literature, these are referred to as modular decompositions, whereas we use that term for general modular decompositions. For a prime node μ in the canonical modular decomposition of G, for every child ν of μ, $L(\nu)$ is a maximal module in $G[L(\mu)]$ and for every maximal module M in $G[L(\mu)]$ there exists a child ν of μ with $L(\nu) = M$. McConnell and Spinrad showed that the canonical modular decomposition can be computed in $O(|V| + |E|)$ time [27]. Let μ be a node in a modular decomposition for G. A *μ-set* U is a subset of $L(\mu)$ that contains for each child λ of μ at most one leaf in $L(\lambda)$. If U contains for every child λ of μ a vertex in $L(\lambda)$, we call it *maximal*.

Lemma 1 *Let T be a modular decomposition of a graph G. After a linear-time preprocessing we can assume that each node of T is annotated with its quotient graph. Moreover, the following queries can be answered in $O(1)$ time:*

1. *Given a non-root node ν of T, find the vertex of the quotient graph of ν's parent that corresponds to ν.*
2. *Given a vertex v in a quotient graph $G[\mu]$, find the child of μ that corresponds to v.*
3. *Given an edge e of G, determine* rep(e)*, the quotient graph that contains* rep(e)*, and which endpoint of* rep(e) *corresponds to which endpoint of e.*

Additionally, given a node μ in T one can find a maximal μ-set U in $O(|U|)$ time.

Proof We focus on constructing the quotient graphs. The queries can be answered by suitably storing pointers during the construction.

For every node μ in T we initiate the quotient graph with one vertex for each child of μ and equip the children of μ and their corresponding vertices with pointers so that queries 1) and 2) can be answered in $O(1)$ time.

Next, we compute the edges of the quotient graphs. The difficulty here is to find for each edge uv in a quotient graph $G[\mu]$ the children λ_u, λ_v of μ with $u \in L(\lambda_u)$ and $v \in L(\lambda_v)$.

For each node μ of T we compute a list L_μ that contains all edges uv of G with $\mathrm{lca}_T(u, v) = \mu$. Namely, we use the lowest-common-ancestor data structure for static trees of Harel and Tarjan [28], to compute $\mathrm{lca}_T(u, v)$ for each edge uv and add uv to $L_{\mathrm{lca}(u,v)}$. Afterwards, we perform a bottom-up traversal of the inner nodes of T that maintains for each leaf v of T the *processed root* $r(v) = \mu$, where μ is the highest already-processed node of T with $v \in L(\mu)$.

Initially, we set $r(v) = v$ for all leaves of T, and we mark all leaves as processed. When processing a node μ, we determine the edges of $G[\mu]$ as follows. We traverse the list L_μ and for each edge $uv \in L_\mu$, we determine the children $r(u)$ and $r(v)$ of μ. From this, we determine the corresponding vertices of $G[\mu]$ and add an edge with a pointer to uv between them. This may create multi-edges. We find those by sorting the incidence list of each vertex by the number of the other incident node in linear time using radix sort [29] and an arbitrary enumeration of $V(G[\mu])$. We then replace all parallel edges between two vertices with a single edge and annotate it with all pointers

of the merged edges. For each pointer from an edge e in $G[\mu]$ to a represented edge uv, we then annotate uv with a pointer to e.

Afterwards, we update $r(v)$ for all $v \in L(\mu)$ to μ and mark μ as processed. To maintain the processed roots of all leaves, we employ a union-find data structure, which initially contains one singleton set for each leaf and when a node μ has been processed, we equip it with a maximal μ-set containing an arbitrarily chosen vertex from every set associated with a child of μ and afterwards unite the sets associated with its children. Since the union-find tree is known in advance (it corresponds to T), the union-find operations can be performed in amortized $O(1)$ time [30].

We observe that the total size of all lists L_μ is $O(m)$ and moreover T has at most $2n - 1$ nodes. Therefore the whole preprocessing runs in linear time. □

Let μ be a node in a modular decomposition T of G and let $\mu_1\mu_2$ be an edge in $G[\mu]$. Let $\vec{T}[G]$ denote an assignment of directions to all edges in $G[\mu]$ for every node μ in T. Such a $\vec{T}[G]$ is *transitive* if it is transitive on every $G[\mu]$. We obtain an orientation of G from an orientation $\vec{T}[G]$ of the quotient graphs of T as follows. Every undirected edge uv in E with $\text{rep}(uv) = \mu_1\mu_2 \in \vec{T}[G]$, is directed from u to v. We say that T *represents* an orientation O of G, if there exists an orientation $\vec{T}[G]$ of the quotient graphs of T that gives us O. We denote the set of all transitive orientations of G represented by T by $\text{to}(T)$. We get an orientation of the quotient graphs of T from an orientation of G, if for each oriented edge $\mu_1\mu_2$, all edges represented by $\mu_1\mu_2$ are oriented from $L(\mu_1)$ to $L(\mu_2)$. Let T now be the canonical modular decomposition of G. Then T represents exactly the transitive orientations of G [26]. It follows that G is a comparability graph if and only if T can be oriented transitively.

If G is a comparability graph, every prime quotient graph $G[\mu] = (V_\mu, E_\mu)$ has exactly two transitive orientations, one the reverse of the other [24], and with the algorithm by McConnell and Spinrad [27] we can compute one of them in $O(|V_\mu| + |E_\mu|)$ time. Hence the time to compute the canonical modular decomposition in which every prime node is labeled with a corresponding transitive orientation is $O(|V|+|E|)$.

3 Transitive Orientation Extension

The *partial orientation extension problem* for comparability graphs OrientExt is to decide for a comparability graph G with a partial orientation W, i.e. an orientation of some of its edges, whether there exists a transitive orientation O of G with $W \subseteq O$. The notion of partial orientations and extensions extends to modular decompositions. We get a partial orientation of the quotient graphs of T from W such that exactly the edges that represent at least one edge in W are oriented and all edges in W that are represented by the same oriented edge $\mu_1\mu_2$, are directed from $L(\mu_1)$ to $L(\mu_2)$.

Lemma 2 *Let T be the canonical modular decomposition of a comparability graph G and let W be a partial orientation of G that gives us a partial orientation P of the quotient graphs of T. Then W extends to a transitive orientation of G if and only if P extends to a transitive orientation of the quotient graphs of T.*

Proof Let O be a transitive orientation of G that extends W. Let $\mu_1\mu_2$ be an oriented edge in P. Then $\mu_1\mu_2$ represents an oriented edge uv in W. Then uv is an oriented

edge in O and $\mu_1\mu_2$ is in the transitive orientation $\vec{T}[G]$ of the quotient graphs of T we get from O. Hence, $\vec{T}[G]$ extends P.

Conversely, let $\vec{T}[G]$ be a transitive orientation of the quotient graphs of T that extends P. Let uv be an oriented edge in W. Then uv is represented by an oriented edge $\mu_1\mu_2$ in P. Then $\mu_1\mu_2$ is an oriented edge in $\vec{T}[G]$ and uv is in the transitive orientation O of G we get from $\vec{T}[G]$. Hence, O extends W. □

To solve OrientExt efficiently we confirm that the partial orientation actually gives us a partial orientation P of the quotient graphs of the canonical modular decomposition T. Otherwise we can reject. By Lemma 2 we now just need to check for each node μ of T whether P can be extended to $G[\mu]$. To this end, we use that μ is empty, complete or prime. Since transitive orientations of cliques are total orders and prime graphs have at most two transitive orientations, the existence of an extension can easily be decided in each case.

Theorem 1 OrientExt *can be solved in linear time.*

Proof Let W be the given partial orientation of a comparability graph $G = (V, E)$. After the linear-time preprocessing of Lemma 1, we can compute the partial orientation P of the quotient graphs of T we get from W in linear time by determining $\text{rep}(uv)$ for every edge $uv \in W$. If P does not exist, then there is an edge $\mu_1\mu_2$ in a quotient graph that represents an edge $e_1 \in W$ oriented from $L(\mu_1)$ to $L(\mu_2)$ and an edge $e_2 \in W$ oriented from $L(\mu_2)$ to $L(\mu_1)$. Then W can not be extended to a transitive orientation of G since in any orientation represented by T the edges e_1, e_2 are both oriented in the same direction between $L(\mu_1)$ and $L(\mu_2)$. Hence, we can reject in this case.

Otherwise, to solve OrientExt for G, it suffices to solve OrientExt for every quotient graph in the canonical modular decomposition T of G with the partial orientation from P by Lemma 2. Let μ be a node in T. We distinguish cases based on the type of μ. If μ is empty, nothing needs to be done. If μ is complete, the problem of extending the partial orientation of $G[\mu]$ is equivalent to the problem of finding a total order of the nodes of $G[\mu]$ that respects P. This can be done via topological sorting in linear time. If μ is prime, $G[\mu]$ has exactly two transitive orientations, where one is the reverse of the other. Therefore we check in linear time whether one of these orientations of $G[\mu]$ is an extension of the partial orientation of $G[\mu]$. Otherwise, no transitive extension exists.

Since we can compute T in $O(|V| + |E|)$ time, in total we can decide whether the partial orientation W is extendible in the same time. we get a corresponding transitive orientation of G by the extension of P and can also be computed in the same time. □

4 Sunflower Orientations

The idea to solve SimOrient* is to obtain for each input graph G_i a restricted modular decomposition of the shared graph H that represents exactly those transitive orientations of H that can be extended to G_i. The restricted modular decompositions can be expressed by constraints for the canonical modular decomposition of H. These

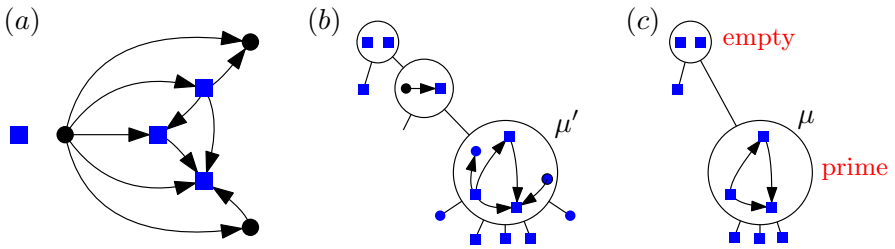

Fig. 3 **a** A graph G with an induced subgraph H (blue square vertices). **b** A modular decomposition T of G with a transitive orientation. **c** The restricted modular decomposition $(T|_H, D)$ of H derived from the transitive orientation in (**b**). Note that $H[\mu]$ is a clique but μ is labeled prime (Color figure online)

constraints are then combined to represent all transitive orientations of H that can be extended to each input graph G_i. With this the solution is straightforward.

Let H be a comparability graph. Then we define a *restricted modular decomposition* (T, D) of H to be a tuple where T is a modular decomposition of H where every node is labeled as complete, empty or prime, such that for every node μ labeled as complete or empty, $H[\mu]$ is complete or empty, respectively, and D is a function that assigns to each prime labeled node μ a transitive orientation D_μ, called *default orientation*. In the following, when referring to the type of a node μ in a restricted modular decomposition, we mean the type that μ is labeled with. A transitive orientation of (T, D), is a transitive orientation of the quotient graphs of T where every prime node μ has orientation D_μ or its reversal D_μ^{-1}. Let $\text{to}(T, D)$ denote the set of transitive orientations of H we get from transitive orientations of (T, D). We say (T, D) *represents* these transitive orientations. Note that for the canonical modular decomposition B of H we have $\text{to}(T, D) \subseteq \text{to}(B)$.

Let G be a comparability graph with an induced subgraph H. A modular decomposition T of G gives us a restricted modular decomposition $(T|_H, D)$ of H as follows; see Fig. 3.

We obtain $T|_H$ from T by (i) removing all leaves that do not correspond to a vertex of H and then (ii) iteratively contracting all inner nodes of degree at most 2. With a bottom-up traversal we can compute $T|_H$ in time linear in the size of T. A node μ in $T|_H$ *stems from* $\text{lca}_T(L(\mu))$. Every node $\mu \in T|_H$ that stems from a prime node $\mu' \in T$ we label as prime and set D_μ to a transitive orientation of $G[\mu']$ restricted to the edges of H. The remaining nodes are labeled according to the type of their quotient graph. Note that $H[\mu]$ is isomorphic to an induced subgraph of $G[\mu']$.

Lemma 3 *Let T be a modular decomposition of a graph G with an induced subgraph H. Then $\text{to}(T|_H, D)$ is the set of orientations of H extendable to transitive orientations of G.*

Proof Let O_H be a transitive orientation of H that can be extended to a transitive orientation O_G of G. Then O_G gives us a transitive orientation $\vec{T}[G]$ of the quotient graphs of T that in turn gives us a transitive orientation $\vec{T}|_H$ of $(T|_H, D)$. Since O_G is an extension of O_H and $\vec{T}[G]$ contains $\vec{T}_H[H]$, O_H is the transitive orientation of H we get from $\vec{T}_H[H]$. Hence $(T|_H, D)$ represents O_H.

Conversely, let $\vec{T}_H[H]$ be a transitive orientation of (T_H, D). For any node μ in $T|_H$ let $\mu' = \text{lca}_T(L(\mu))$. Recall that $H[\mu]$ is isomorphic to an induced subgraph of $G[\mu']$. We already know that $G[\mu']$ is either empty, complete or prime. If $G[\mu']$ is empty, $H[\mu]$ and the corresponding transitive orientation are also empty. If $G[\mu']$ is complete, then $H[\mu]$ is also complete and any transitive orientation of $H[\mu]$ can be extended to a transitive orientation of $G[\mu']$. If $G[\mu']$ is prime, by construction $H[\mu]$ is also labeled as prime with a default orientation D_μ given by a transitive orientation $D_{\mu'}$ of $G[\mu']$. Hence $\vec{T}_H[H]$ either contains $D_{\mu'} \cap E(H[\mu]) = D_\mu$ or $D_{\mu'}^{-1} \cap E(H[\mu]) = D_\mu^{-1}$. Thus $\vec{T}_H[H]$ can be mapped to T and be extended to a transitive orientation $\vec{T}[G]$ of the quotient graphs of T. Then $\vec{T}[G]$ gives us a transitive orientation O_G of G since T represents exactly the transitive orientations of G. Let O_H be O_G restricted to H. Then by construction O_H equals the orientation of H we get from $\vec{T}_H[H]$. Thus $\vec{T}_H[H]$ gives us a transitive orientation of H. □

Consider the canonical modular decompositions $T^1, \ldots, T^r$ of the input graphs $G_1, \ldots, G_r$, and let $(T^1|_H, D_1), \ldots, (T^r|_H, D_r)$ be the corresponding restricted modular decompositions. Then we are interested in the intersection $\text{to}(G_1, \ldots, G_r) = \bigcap_{i=1}^r \text{to}(T^i|_H, D_i)$ since it contains all transitive orientations of H that can be extended to all input graphs. However, the trees $T^1|_H, \ldots, T^r|_H$ have different shapes, which makes it difficult to compute a representation of $\text{to}(G_1, \ldots, G_r)$ directly. Instead, we describe the sets of transitive orientations $\text{to}(T^1|_H, D_1), \ldots, \text{to}(T^r|_H, D_r)$ (whose intersection is simple to compute) with constraints on the canonical modular decomposition B of H.

Let (T, D) be a restricted modular decomposition of H and let B be the canonical modular decomposition of H. We collect constraints on the orientations of individual nodes μ of B that are imposed by $\text{to}(T, D)$. Afterwards we show that the established constraints are sufficient to describe $\text{to}(T, D)$. If μ is empty, then $H[\mu]$ is empty and has a unique transitive orientation, which requires no constraints. The other types of μ are discussed in the following two sections.

4.1 Constraints for Prime Nodes

In this section we observe that prime nodes in B correspond to prime nodes in (T, D) and that the dependencies between their orientations can be described with 2-SAT formulas. Recall that the transitive orientation of a prime comparability graph is unique up to reversal. We consider each prime node $\mu \in B$ equipped with a transitive orientation D_μ, also called a *default orientation*. All default orientations for B can be computed in linear time [27].

Lemma 4 *Let $\mu \in B$ be prime. Then $\mu' = \text{lca}_T(L(\mu))$ is prime, every μ-set is a μ'-set, and for any edge uv with $\text{rep}_B(uv) \in H[\mu]$ we have $\text{rep}_T(uv) \in H[\mu']$.*

Proof We first show that every μ-set is also a μ'-set. Assume there is a μ-set U that is not a μ'-set. Since $U \subseteq L(\mu) \subseteq L(\mu')$, there exist two vertices $u \neq v \in U$ with $\lambda = \text{rep}_{\mu'}(u) = \text{rep}_{\mu'}(v)$. From $L(\lambda)$ being a module of $H[L(\mu')]$ and $L(\mu) \subseteq L(\mu')$ it follows that $X = L(\lambda) \cap L(\mu)$ is a module of $H[L(\mu)]$. By the definition of μ', we

have $X \subsetneq L(\mu)$. Since μ is prime, there is a child v of μ with $u, v \in X \subseteq L(v)$ contradicting U being a μ-set.

As a direct consequence of μ-sets being μ'-sets, we also have for any edge uv with $\text{rep}_B(uv) \in H[\mu]$ that $\text{rep}_T(uv) \in H[\mu']$ since $\{u, v\}$ is a μ-set. Now let U be a maximal μ-set. Then $H[U]$ is isomorphic to $H[\mu]$ and thus prime. Since U is also a μ'-set and the subgraph of $H[\mu']$ induced by the vertices representing U is isomorphic to $H[U]$, μ' is neither empty, nor complete and thus prime. □

For a modular decomposition T' of H with a node λ and $O \in \text{to}(T')$ let $O_{\downarrow\lambda}$ denote the orientation of the quotient graph $H[\lambda]$ we get from O. Note that for a transitive orientation D_λ of $H[\lambda]$ and a λ-set U each orientation $O \in \text{to}(T')$ with $O_{\downarrow\lambda} = D_\lambda$ gives us the same orientation on $H[U]$. We say that D_λ *induces* this orientation on $H[U]$.

Let $\mu \in B$ be prime, let $\mu' = \text{lca}_T(L(\mu))$ and let U be a maximal μ-set (and by Lemma 4 a μ'-set). We set $D^\delta_{\mu'} = D_{\mu'}$ if $D_\mu, D_{\mu'}$ induce the same transitive orientation on the prime graph $H[U]$ and we set $D^\delta_{\mu'} = D^{-1}_{\mu'}$ if the induced orientations are the reversal of each other. Note that $D^\delta_{\mu'}$ does not depend on the choice of U. From the definition of $D^\delta_{\mu'}$ and the observation that $O_{\downarrow\mu}, O_{\downarrow\mu'}$ are both determined by O restricted to $H[U]$ we directly get the following lemma.

Lemma 5 *For $O \in to(T, D)$ we have $O_{\downarrow\mu} = D_\mu \Leftrightarrow O_{\downarrow\mu'} = D^\delta_{\mu'}$.*

We express the choice of a transitive orientation for a prime node μ by a Boolean variable x_μ that is `true` for the default orientation and `false` for the reversed orientation.

According to Lemma 5 we set ψ_μ to be $x_\mu \leftrightarrow x_{\mu'}$ if $D^\delta_{\mu'} = D_{\mu'}$ and $x_\mu \nleftrightarrow x_{\mu'}$ if $D^\delta_{\mu'} = D^{-1}_{\mu'}$. Note that for a prime node $\mu' \in T$ there may exist more than one prime node μ in B such that $\mu' = \text{lca}_T(L(\mu))$, and we may hence have multiple prime nodes that are synchronized by these constraints. We describe these dependencies with the formula $\psi = \bigwedge_{\mu \in P(B)} \psi_\mu$. With the above meaning of variables, any choice of orientations for the prime nodes of B that can be induced by T necessarily satisfies ψ. With Lemma 1 we can compute ψ efficiently.

Lemma 6 *We can compute ψ in $O(|V(H)| + |E(H)|)$ time.*

Proof Let μ be a prime node in B and let $\mu' = \text{lca}_T(L(\mu))$. By Lemma 1 we can compute a maximal μ-set U in constant time after a linear-time preprocessing. By Lemma 4 we have for every edge uv with $\text{rep}_B(uv) \in H[\mu]$ that $\text{rep}_T(uv) \in H[\mu']$. Hence we can find μ' in T by determining $\text{rep}_T(uv)$ for an arbitrary edge uv with $u, v \in U$, which by Lemma 1 takes constant time. For an arbitrary oriented edge $e \in O$ we check in constant time whether $e \in D_\mu$ or $e \in D^{-1}_\mu$ and add the clause $x_\mu \leftrightarrow x_{\mu'}$ or $x_\mu \nleftrightarrow x_{\mu'}$, respectively. Doing this for every prime node in total takes time linear in the size of T. □

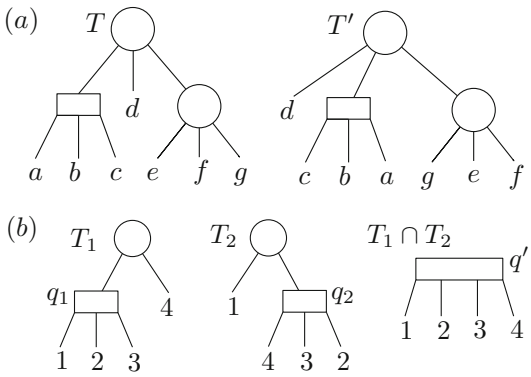

Fig. 4 **a** Two equivalent PQ-trees T, T' with $fr(T) = abcdefg$ and $fr(T') = dcbagef$. **b** The Q-node q' in $T_1 \cap T_2$ contains q_1 forwards and q_2 backwards

4.2 Constraints for Complete Nodes

Next we consider the case where μ is complete. The edges represented in $H[\mu]$ may be represented by edges in more than one quotient graph in T, each of which can be complete or prime. Depending on the type of the involved quotient graphs in T we get new constraints for the orientation of $H[\mu]$.

Note that choosing a transitive orientation of $H[\mu]$ is equivalent to choosing a linear order of the vertices of $H[\mu]$. As we will see, each node ν of T that represents an edge of $H[\mu]$ imposes a consecutivity constraint on a subset of the vertices of $H[\mu]$. Therefore, the possible orders can be represented by a *PQ-tree* that allows us to represent all permissible permutations of the elements of a set U in which certain subsets $S \subseteq U$ appear consecutively.

PQ-trees were first introduced by Booth and Lueker [31, 32]. A PQ-tree T over a finite set U is a rooted tree whose leaves are the elements of U and whose internal nodes are labeled as P- or Q-nodes. A P-node is depicted as a circle, a Q-node as a rectangle; see Fig. 4. Two PQ-trees T and T' are *equivalent*, if T can be transformed into T' by arbitrarily permuting the children of arbitrarily many P-nodes and reversing the order of arbitrarily many Q-nodes; see Fig. 4a. A transformation that transforms T into an equivalent tree T' is called an *equivalence transformation*. The *frontier* $\mathrm{fr}(T)$ of a PQ-tree T is the order of its leaves from left to right. The tree T *represents* the frontiers of all equivalent PQ-trees. The PQ-tree that does not have any nodes is called the *null tree*.

Let T_1, T_2 be two PQ-trees over a set U. Their *intersection* $T = T_1 \cap T_2$ is a PQ-tree that represents exactly the linear orders of U represented by both T_1 and T_2. It can be computed in $O(|U|)$ time [31].

For every Q-node q in T_1 node $q' = \mathrm{lca}_T(L(q))$ is also a Q-node. We say that q' *contains q forwards*, if $T_1[q]$ can be transformed by an equivalence transformation that does not reverse q into a PQ-tree T' such that $\mathrm{fr}(T[q])$ contains $\mathrm{fr}(T[q'])$; see Fig. 4b. Else q' *contains q backwards*. Similarly every Q-node in T_2 is contained in exactly one Q-node in T (either forwards or backwards). Haeupler et al. [33] showed that one can modify Booth's algorithm such that given two PQ-trees T_1, T_2 it not only

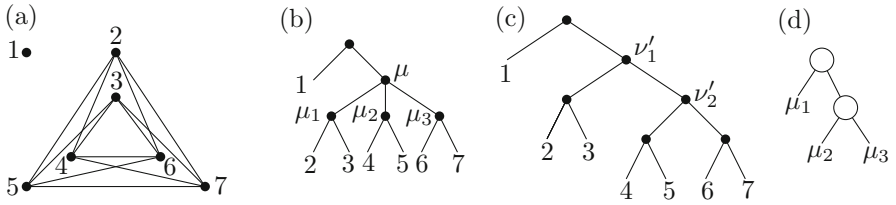

Fig. 5 **a** A graph H. **b** The canonical modular decomposition B of H. **c** A restricted modular decompositions T of H. **d** The PQ-tree S_μ. The active nodes of T with regard to the μ-set $\{2, 4, 6\}$ are ν'_1, ν'_2, 2, 4 and 6

outputs $T_1 \cap T_2$ but also for every Q-node in T_1, T_2 which Q-node in T contains it and in which direction.

Lemma 7 *Let $T_1, \ldots, T_k$ be PQ-trees over a set U. Then we can compute their intersection $T = \bigcap_{i=1}^{k} T_i$ and determine for every Q-node q in $T_1, \ldots, T_k$ the Q-node in T that contains q and in which direction in $O(k \cdot |U|)$ time.*

Proof Let $S_1 = T_1$ and for every $j \in \{2, \ldots, k\}$ let $S_j := S_{j-1} \cap T_i$. To compute $T = S_k$ we stepwise compute $S_j = S_{j-1} \cap T_j$ for every j. During the computation of T we construct a DAG D whose vertices are the Q-nodes of $T_1, \ldots, T_k$ and $S_2, \ldots, S_k$. Initially D contains no edges. For every j we add a directed edge from every Q-node q in T_j and S_{j-1} to the Q-node q' in S_j that contains q. We label the edge with 1 if q' contains q forwards, and with -1 otherwise. By the result of Haeupler et al. this can be done in $O(k \cdot |U|)$ time [33]. Note that by construction every vertex has at most one outgoing edge and for every Q-node q in $T_1, \ldots, T_k$ there is a unique path to the Q-node q' in S_k that contains it. The product of the edge labels along this path is 1 and -1 if q' contains q forwards and backwards, respectively.

To determine for every Q-node q in $T_1, \ldots, T_k$ which Q-node in T contains it and in which direction, we start at the sinks in D and backward propagate for every vertex in D the information which unique sink q' can be reached from it and what is the product of edge labels along the path to q'. This needs $O(k \cdot |U|)$ time since D has $O(k \cdot |U|)$ vertices and edges. □

Let $\mu' = \mathrm{lca}_T(L(\mu))$ for the rest of this section and let $U \subseteq V$ be a maximal μ-set. We call a node of T *active* if it is either a leaf in U or if at least two of its subtrees contain leaves in U. Denote by A the set of active nodes in T and observe that A can be turned into a tree S_μ by connecting each node $\nu' \in A \backslash \{\mu'\}$ to its lowest active ancestor; see Fig. 5. Let now $\vec{B}[H]$ be an orientation of the quotient graphs of B induced by an orientation $\vec{T}[G] \in \mathrm{to}(T, D)$ and consider a node $\nu' \neq \mu'$ of S_μ. Let $X = U \cap L(\nu')$ and let $Y = U \backslash L(\nu')$. Since U is a μ-set of a complete node, any pair of vertices in $X \times Y$ is adjacent. Moreover, for each $y \in Y$, the edges from y to X are all oriented towards X, or they are all oriented towards y, since every node of T that determines the orientation of such an edge contains all vertices of X in a single child. This implies that in the order of the μ-set given by the order of $H[\mu]$, the set $L(\nu')$ is consecutive. Moreover, if ν' is prime, its default orientation $D_{\nu'}$ induces a total order on the active children of ν' that is fixed up to reversal. Hence we turn S_μ into a PQ-tree by first

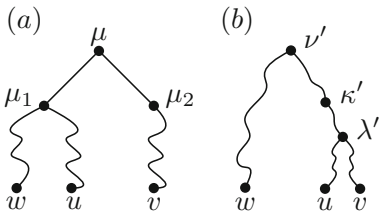

Fig. 6 **a** A subtree of B with root μ. **b** A subtree of T with root ν'

turning all complete nodes into P-nodes and all prime nodes into Q-nodes with the children ordered according to the linear order determined by the default orientation which we call the *initial* order of the Q-node. Finally, we replace each leaf $v \in U$ by the corresponding vertex $\text{rep}_\mu(v)$ of $H[\mu]$; see Fig. 5. As argued above, the linear order of $H[\mu]$ is necessarily represented by S_μ.

We show that tree S_μ is independent from the choice of the maximal μ-set U. We use that any node of T has a laminar relation to the children of μ. More precisely, for every node κ' in T, either all leaves shared by $L(\kappa')$ and $L(\mu)$ are descendants of the same child of μ, or for every child μ_1 of μ with $L(\kappa') \cap L(\mu_1) \neq \emptyset$ we have that $L(\mu_1)$ is completely contained in $L(\kappa')$.

Lemma 8 *For any child μ_1 of μ and any node κ' of $V(T)$ with $L(\kappa') \cap L(\mu_1) \neq \emptyset$ we have $L(\mu_1) \subseteq L(\kappa') \cap L(\mu)$ or $L(\kappa') \cap L(\mu) \subseteq L(\mu_1)$.*

Proof If $L(\kappa')$ contains only leaves of at most one child μ_1 of μ, we have $L(\kappa') \cap L(\mu) \subseteq L(\mu_1)$ and $L(\mu_2) \cap L(\kappa') = \emptyset$ for each remaining child μ_2 of μ. Otherwise, let μ_1, μ_2 be children of μ with leaves $u \in L(\kappa') \cap L(\mu_1)$, $v \in L(\kappa') \cap L(\mu_2)$ and let $\lambda' = \text{lca}_T(u, v)$; see Fig. 6. Note that λ' is a descendant of κ' or $\lambda' = \kappa'$. Assume that there exists a vertex $w \in L(\mu_1) \backslash L(\kappa')$ and let $\nu' = \text{lca}_T(u, w)$. Then κ' is a descendant of ν' since $u \in L(\nu') \cap L(\kappa')$ and $w \in L(\nu') \backslash L(\kappa')$. Note that $\text{rep}_T(vw) = \text{rep}_T(uw) \in H[\nu']$ and $\text{rep}_T(uv) \in H[\lambda']$. Hence, T represents a transitive orientation of H with uv and vw. This contradicts B representing all transitive orientations of H since we have $\text{rep}_B(uv) = \text{rep}_B(wv)$. It follows that $L(\mu_1) \backslash L(\kappa') = \emptyset$ and analogously $L(\mu_2) \backslash L(\kappa') = \emptyset$. This concludes the proof. □

Lemma 9 *Let S^1_μ, S^2_μ be the PQ-trees for two maximal μ-sets U_1, U_2, respectively. Then $S^1_\mu = S^2_\mu$.*

Proof Let ν' be a non-leaf node in T that is active with regard to U_1. Then there exist two vertices $u_1, v_1 \in U_1$ and two distinct children ν'_1, ν'_2 of ν' such that $u_1 \in L(\nu'_1)$ and $v_1 \in L(\nu'_2)$. Let $\mu_1 = \text{rep}_\mu(u_1)$, $\mu_2 = \text{rep}_\mu(v_1)$ and let u_2, v_2 be the vertices in U_2 with $\text{rep}_\mu(u_2) = \mu_1$, $\text{rep}_\mu(v_2) = \mu_2$. We prove the following:

1. $u_2, v_2 \in L(\nu')$
2. $\text{rep}_{\nu'}(u_2) \neq \text{rep}_{\nu'}(v_2)$
3. $\text{rep}_{\nu'}(u_1 v_1) \in D_{\nu'} \Leftrightarrow \text{rep}_{\nu'}(u_2 v_2) \in D_{\nu'}$.

For Statement 1 observe that $u_1 \in L(\nu') \cap L(\mu_1) \neq \emptyset$ and $v_1 \in L(\nu') \cap L(\mu) \backslash L(\mu_1)$. Hence $L(\nu') \cap L(\mu) \not\subseteq L(\mu_1)$ and thus by Lemma 8 $u_2 \in L(\mu_1) \subseteq L(\nu') \cap L(\mu)$. Analogously we get $v_2 \in L(\mu_2) \subseteq L(\nu') \cap L(\mu)$.

For Statement 2 assume $\nu'' = \text{rep}_{\nu'}(u_2) = \text{rep}_{\nu'}(v_2)$. Then we have $u_2 \in L(\nu'') \cap L(\mu_1)$ and $v_2 \in L(\nu'') \cap L(\mu) \setminus L(\mu_1)$. By Lemma 8 we get $u_1 \in L(\mu_1) \subseteq L(\nu'') \cap L(\mu)$, i.e., $\nu'' = \nu'_1$ and similarly $\nu'' = \nu'_2$ in contradiction to $\nu'_1 \neq \nu'_2$.

For Statement 3 note that u_1v_1 and u_2v_2 are indeed represented in ν' by Statement 2 and they are both represented by $\mu_1\mu_2$ in $H[\mu]$.

It follows that the active inner nodes are the same for U_1, U_2 and after replacing the leaves with the children of μ we obtain the same trees. Statement 3 then provides that the ordering of the children of Q-nodes is also the same and S^1_μ, S^2_μ are indeed the same PQ-tree. □

By construction, each Q-node q of S_μ stems from a prime node ν' in T, and the orientation of $T[\nu']$ determines the orientation of q, namely q is reversed if and only if $T[\nu']$ is oriented as $D^{-1}_{\nu'}$. Since a single prime node ν' of T may give rise to Q-nodes in several PQ-trees S_μ, we need to ensure that the orientations of these Q-nodes are either all consistent with the default orientation of $T[\nu']$ or they are all consistent with its reversal. To model this, we introduce a Boolean variable x_q for each Q-node q in one of the PQ-trees with the interpretation that $x_q = \texttt{true}$ if and only if q has its initial order. We require x_q to be equal to the variable that orients the prime node corresponding to q. More precisely, for every prime node ν' in $T[\mu']$ that gives rise to q we add the constraint $(x_{\nu'} \leftrightarrow x_q)$ to χ_μ, where the variable $x_{\nu'}$ is the variable that encodes the orientation of the prime node ν'. We construct a Boolean formula by setting $\chi = \bigwedge_{\mu \in K(B)} \chi_\mu$.

Lemma 10 *We can compute all PQ-trees S_μ and the formula χ in $O(n+m)$ time.*

Proof As a preprocessing we run a DFS on T starting at the root and store for every node ν its discovery-time $\nu.d$, i.e., the timestamp when ν is first discovered, and its finish-time $\nu.f$, i.e., the timestamp after all its neighbors have been examined. We also employ the preprocessing from Lemma 1. We construct all PQ-trees and χ with the following steps.

1. Take a maximal μ-set U_μ for every $\mu \in K(B)$.
2. For every $\mu \in K(B)$ compute the set of active nodes and for every active node compute its parent in S_μ.
3. For every $\mu \in K(B)$ determine for each inner node of S_μ whether it is a P- or a Q-node. If it is a Q-node, determine the linear order of its children, and construct the formula χ_μ.

Step 1 can be done in $O(n)$ time by Lemma 1.

For Step 2, note that each active node is a least common ancestor of two leaves in U_μ. While it is easy to get all active nodes as least common ancestors, getting the edges of S_μ requires more work. Observe that the DFS on T visits the nodes of S_μ in the same order as a DFS on S_μ. Consider S_μ embedded such that the children of each node are ordered from left to right by their discovery-times. This also orders the leaves from left to right by their discovery-times. Let λ be an inner node of S_μ. Let λ_1, λ_2 be two neighboring children of λ with λ_1 to the left of λ_2. Then λ is the least common ancestor of the rightmost leaf in $L(\lambda_1)$ and the leftmost leaf in $L(\lambda_2)$. Hence, each node of S_μ is a least common ancestor for a consecutive pair of leaves.

We add for every node u in a set U_μ a tuple $(\mu, u.d)$ to an initially empty list L. We then sort the tuples in L in linear time using radix sort [29]. In the sorted list, for every $\mu \in K(B)$ all tuples $(\mu, u.d)$ are consecutive and the consecutive sublist is sorted by discovery time.

For $\mu \in K(B)$ let L_μ be a list containing the vertices in U_μ ordered by their discovery time which we get directly from the consecutive sublist of L containing the tuples corresponding to μ. For every pair $u, v \in U_\mu$ adjacent in L_μ we compute $\lambda = \mathrm{lca}_T(u, v)$ using the lowest-common-ancestor data structure for static trees by Harel and Tarjan [28] and insert λ into L_μ between u and v. For a vertex $u \in U_\mu$ its parent in S_μ is the neighbor in L_μ that has a lower position in T. Note that u is a descendent in T of all its neighbors in L_μ. Hence if u has two neighbors in L_μ one of them is a descendent of the other. Thus the parent of u in S_μ is the neighbor with the higher discovery time. Now we remove all vertices in U_μ and possible duplicates of the remaining nodes from L_μ. Note that still every λ is a descendent in T of its neighbors in L_μ. Hence we iteratively choose a node λ in L_μ whose discovery time is higher than the discovery time of its neighbors, compute its parent in S_μ by comparing the discovery times of its neighbors with each other and remove λ from L_μ.

In Step 3, we turn each active node that stems from a complete node into a P-node and each active node that stems from a prime node into a Q-node. For a Q-node q that stems from a prime node ν, we determine the linear order of its children as follows. Take the set X of vertices of $H[\nu]$ that correspond to children of μ in S_μ, determine the orientation of the complete graph on X induced by D_ν and sort it topologically. In total this take $O(n+m)$ time for all active nodes in all PQ-trees. Using the information computed up to this point, it is straightforward to output the formula χ. □

Finally, we combine the constraints from the complete nodes with the constraints from the prime nodes by setting $\varphi_T = \psi \wedge \chi$. The formula φ_T allows us to describe a restricted set of transitive orientations of G. We define $S_T = \{S_\mu | \mu \in K(B)\}$. The canonical modular decomposition (B, S_T, φ_T) of H where every complete node is labeled with the corresponding PQ-tree together with φ_T we call a *constrained* modular decomposition.

We say that a transitive orientation O of H *induces a variable assignment satisfying* φ_T if it induces an assignments of the variables corresponding to prime nodes in B and Q-nodes such that φ_T is satisfied for an appropriate assignment for the variables corresponding to prime nodes in T. Let $\mathrm{to}(B, S_T, \varphi_T)$ denote the set containing all transitive orientations $O \in \mathrm{to}(B)$ where for every complete node $\mu \in B$ the order $O_{\downarrow\mu}$ corresponds to a total order represented by S_μ and that induces a variable assignment that satisfies φ_T.

4.3 Correctness

We now show that $\mathrm{to}(B, S_T, \varphi_T) = \mathrm{to}(T, D)$. To this end we use that Lemma 4 allows to find for an edge uv that is represented in a prime node μ of B the prime node $\mu' = \mathrm{lca}_T(L(\mu))$ of T where it is represented. This allows us to establish the identity of certain nodes. The following lemma does something similar for complete nodes.

Lemma 11 *Let uv, wx be edges of H represented in complete nodes μ, ν of B and by the same edge in a complete node of T. Then $\mu = \nu$.*

Proof Let μ be the node in B such that $\text{rep}_B(uv) \in H[\mu]$ and let ν be the node in B such that $\text{rep}_B(wx) \in H[\nu]$. Let ω' be the node in T with $\text{rep}_T(uv) \in H[\omega']$ and $\text{rep}_T(wx) \in H[\omega']$. Assume $\mu \neq \nu$. Then one of them is the ancestor of the other or they are both distinct from $\lambda = \text{lca}_B(\mu, \nu)$. First consider the case that μ is an ancestor of ν. Then μ has a child μ_1 such that $L(\nu) \subseteq L(\mu_1)$. Note that $\text{rep}_\mu(u) \neq \text{rep}_\mu(v)$, hence we have $\text{rep}_\mu(u) \neq \mu_1$ or $\text{rep}_\mu(v) \neq \mu_1$. Without loss of generality assume $\text{rep}_\mu(u) \neq \mu_1$.

Since $\{w, x\} \subseteq L(\omega') \cap L(\mu_1)$ and $u \in L(\omega') \cap L(\mu) \backslash L(\mu_1)$ we have $L(\mu_1) \subseteq L(\omega')$ by Lemma 8 and analogously it follows that $L(\text{rep}_\mu(u)) \subseteq L(\omega')$. Since $\text{rep}_T(wx) = \text{rep}_T(uv) \in H[\omega']$ it is $\omega' = \text{lca}_T(L(\mu_1))$.

Assume that μ_1 is prime. Then by Lemma 4, ω' is prime which is a contradiction to the assumption that ω' is complete. Hence μ_1 must be complete but as B is the modular decomposition of H, no two adjacent nodes in B are complete. Thus μ is not an ancestor of ν and analogously we get that ν is not an ancestor of μ.

It remains to consider the case that $\nu \neq \lambda \neq \mu$. Let λ_1 be the child of λ such that $L(\mu) \subseteq L(\lambda_1)$ and let λ_2 be the child of λ such that $L(\nu) \subseteq L(\lambda_2)$. Again λ has to be complete since otherwise by Lemma 4 ω' would be prime which is a contradiction. By Lemma 8 we have that $L(\lambda_1) \cup L(\lambda_2) \subseteq L(\omega')$.

Assume that λ_1 is prime. Then by Lemma 4, ω' is prime which leads to a contradiction. Hence λ_1 must be complete but again as B is the modular decomposition of H, no two adjacent nodes in B are complete. □

Theorem 2 *Let B be the canonical modular decomposition for a graph H and let T be a restricted modular decomposition for H. Then $to(B, S_T, \varphi_T) = to(T, D)$ and $to(B, S_T, \varphi_T)$ can be computed in time that is linear in the size of H.*

Proof Let $O_H \in \text{to}(T, D)$ and let $\vec{B}[H]$ be the orientation of the quotient graphs of B inducing O_H. Then we have already seen that it is necessary that every complete node μ in B is oriented according to a total order represented by S_μ and that O_H induces a variable assignment that satisfies φ_T. Hence $O_H \in \text{to}(B, S_T, \varphi_T)$.

Conversely, let $O_H \in \text{to}(B, S_T, \varphi_T)$ and assume $O_H \notin \text{to}(T, D)$. Then O_H is either not represented by T or does not induce D. I.e., O_H contains two directed edges uv, wx with $\text{rep}_T(uv)$ and $\text{rep}_T(wx)$ in the same quotient graph $H[\omega']$, such that $\text{rep}_T(uv) = \text{rep}_T(xw)$, or ω' is prime and $\text{rep}_T(uv) \in D_{\omega'}$ but $\text{rep}_T(wx) \in D_{\omega'}^{-1}$. Note that if ω' is prime, then the first case implies the second one. Let $\mu = \text{lca}_B(u, v)$ and $\nu = \text{lca}_B(w, x)$. We distinguish cases based on the types of μ and ν. Let $\mu' = \text{lca}_T(L(\mu))$, $\nu' = \text{lca}_T(L(\nu))$ and let $\vec{B}[H]$ be the orientation of the quotient graphs of B that induces O_H. Without loss of generality, assume $\text{rep}_B(uv) \in D_\mu$ and $\text{rep}_B(wx) \in D_\nu$.

Case 1: μ and ν are both prime. By Lemma 4 we have that $\mu' = \omega' = \nu'$ is prime. By construction, φ_T enforces that uv, wx are either represented in $D_{\omega'}$ or in $D_{\omega'}^{-1}$. Hence, this case does not occur.

Case 2: μ is prime and ν is complete. By Lemma 4 we have that $\mu' = \omega'$ is prime. Let U be a ν-set containing w and x. Since $\text{rep}_T(wx) \in H[\omega']$, node ω' is active with respect to U. Hence the PQ-tree S_ν contains a Q-node q that stems from ω'.

By construction φ_T contains the constraints $(x_{\omega'} \leftrightarrow x_q)$ and $(x_{\omega'} \leftrightarrow x_\mu)$ but $\vec{B}[H]$ induces $x_\mu = \texttt{true}$, $x_q = \texttt{false}$. Hence the variable assignment induced by $\vec{B}[H]$ does not satisfy φ_T and thus $O_H \notin \text{to}(B, S_T, \varphi_T)$.

Case 3: μ is complete and ν is prime. Similar to Case 2.

Case 4: μ, ν are both complete. Here we further distinguish two subcases depending on the type of ω'. First assume that ω' is prime. Let V_1' be a μ-set containing u, v and let V_2' be a ν-set containing w, x. Since $\text{rep}_T(uv)$ and $\text{rep}_T(wx)$ are edges in $H[\omega']$, node ω' is active with respect to both V_1', V_2'. Hence S_μ, S_ν contain Q-nodes q_1, q_2 stemming from ω'. By construction φ_T contains the constraints $(x_{\omega'} \leftrightarrow x_{q_1})$ and $(x_{\omega'} \leftrightarrow x_{q_2})$, but $\vec{B}[H]$ induces $x_{q_1} = \texttt{true}$, $x_{q_2} = \texttt{false}$. Hence the variable assignment induced by O_H does not satisfy φ_T and thus $O_H \notin \text{to}(B, S_T, \varphi_T)$.

It remains to consider the case that ω' is complete. By Lemma 11 we have $\mu = \nu$. Since ω' is not prime, we must have $\text{rep}_T(uv) = \text{rep}_T(xw)$ by assumption. Let U be a μ-set. Since $\text{rep}_T(uv) = \text{rep}_T(xw) \in H[\omega']$, node ω' is active with respect to U and $\text{rep}_{\omega'}(u) = \text{rep}_{\omega'}(x), \text{rep}_{\omega'}(v) = \text{rep}_{\omega'}(w)$. Hence S_μ contains a P-node that stems from ω' and demands a total order of the children of μ where $\text{rep}_\mu(u)$, $\text{rep}_\mu(x)$ are either both smaller than both $\text{rep}_\mu(v)$,$\text{rep}_\mu(w)$, or both $\text{rep}_\mu(u)$,$\text{rep}_\mu(x)$ are greater than both $\text{rep}_\mu(v)$, $\text{rep}_\mu(w)$. Since $\vec{B}[H]$ induces $\text{rep}_\mu(u) < \text{rep}_\mu(v)$ but $\text{rep}_\mu(x) > \text{rep}_\mu(w)$, $H[\mu]$ is not oriented according to a total order represented by S_μ and thus $O_H \notin \text{to}(B, S_T, \varphi_T)$.

By Lemmas 6 and 10 the formula $\varphi_T = \psi \wedge \chi$ and S_T can be computed in linear time. □

Let T be a modular decomposition of a graph G with an induced subgraph H. Let B be the canonical modular decomposition of H and let $T|_H$ be the restricted modular decomposition for H we get from T. From Lemma 3 and Theorem 2 we directly get the following corollary.

Corollary 1 *The set $\text{to}(B, S_{T|_H}, \varphi_{T|_H})$ contains exactly those transitive orientations of H that can be extended to a transitive orientation of G.*

Let $(B, S^1, \varphi_1), \ldots, (B, S^r, \varphi_r)$ be constrained modular decompositions for H. Let $S_\mu = \bigcap_{i=1}^r S_\mu^i$ and $S = \{S_\mu | \mu \in K(B)\}$. The intersection (B, S, φ) of $(B, S^1, \varphi_1), \ldots, (B, S^r, \varphi_r)$ is the constrained modular decomposition of H where every complete node μ is labeled with the PQ-tree S_μ equipped with the 2-SAT-formula $\varphi = (\bigwedge_{i=1}^r \varphi_i) \wedge \varphi'$ where φ' synchronizes the Q-nodes in the S_μ's with the Q-nodes in the S_μ^i's as follows. Recall that for every Q-node q in a tree S_μ^i there exists a unique Q-node q' in S_μ that contains q; either forward or backward. For every $i \in \{1, \ldots, r\}$, every $\mu \in K(B)$ and every Q-node q in S_μ^i we determine the Q-node q' in S_μ that contains q and add the clause $(x_q \leftrightarrow x_{q'})$ if q has its forward orientation in q' and $(x_q \nleftrightarrow x_{q'})$ otherwise.

Lemma 12 *It is $\text{to}(B, S, \varphi) = \bigcap_{i=1}^r \text{to}(B, S^i, \varphi_i)$ and we can compute (B, S, φ) in linear time from $\{(B, S^i, \varphi_i) | 1 \le i \le r\}$.*

Proof Let $O_H \in \bigcap_{i=1}^r \text{to}(B, S^i, \varphi_i)$ and let $\vec{B}[H]$ be the orientation of the quotient graphs of B that induces O_H. Then for every $i \in \{1, \ldots, r\}$ the variable assignment

induced by O_H satisfies φ_i and for every complete node μ in B, the total order $O_{H\downarrow\mu}$ is represented by S^i_μ, hence it is also represented by S_μ. Let $i \in \{1, \ldots, r\}$ and let q be a Q-node in S^i_μ and let q' be the Q-node in S_μ containing q. Without loss of generality assume q' contains q forwards and $\vec{B}[H]$ induces $x_q = \texttt{true}$ with respect to (B, S^i, φ^i). Then $\vec{B}[H]$ induces $x_{q'} = \texttt{true}$ with respect to (B, S, φ) and φ' contains the clause $(x_{q'} \leftrightarrow x_q)$ for x_q. Hence φ' is satisfied in any extension of an assignment of variables corresponding to Q-nodes in $S_\mu, S^1_\mu, \ldots, S^r_\mu$ induced by $\vec{B}[H]$. Hence O_H induces a variable assignment that satisfies φ and thus $O_H \in \text{to}(B, S, \varphi)$.

Conversely let $O_H \in \text{to}(B, S, \varphi)$ and let $\vec{B}[H]$ be the orientation of the quotient graphs of B that induces O_H. Then for every complete node $\mu \in B$, the total order $O_{H\downarrow\mu}$ is represented by S and thus by S^i_μ for $i \in \{1, \ldots, r\}$. Further $\vec{B}[H]$ induces assignments of the variables corresponding to the prime nodes in B and to the Q-nodes in S. These induced variable assignments can be extended to a solution of φ. Let $i \in \{1, \ldots, r\}$ and let q be a Q-node in S^i_μ such that q' is the Q-node in S_μ containing q. Without loss of generality assume q' contains q forwards and $\vec{B}[H]$ induces $x_{q'} = \texttt{true}$ with respect to (B, S, φ). Then φ contains the clause $(x_{q'} \leftrightarrow x_q)$ and hence $x_q = \texttt{true}$ in any solution of φ that is an extension of an assignment of variables corresponding to Q-nodes in S_μ induced by $\vec{B}[H]$. Since for every complete node μ in B, the total order $O_{H\downarrow\mu}$ is also represented by S^i_μ, $\vec{B}[H]$ also induces assignments of the variables corresponding to Q-nodes in S^i_μ. Since q' contains q forwards $\vec{B}[H]$ must induce $x_q = \texttt{true}$ as well with respect to (B, S^i, φ_i). Hence the variable assignment induced by $\vec{B}[H]$ and S^i_μ satisfies φ_i and thus $O_H \in \bigcap_{i=1}^r \text{to}(B, S^i, \varphi_i)$.

It remains to show the linear runtime. Let $G_1 = (V_1, E_1), \ldots, G_r = (V_r, E_r)$ with $n_i = |V_i|$ and $m_i = |E_i|$ for all $1 \leq i \leq r$ and let $n = \sum_{i=1}^r n_i$, $m = \sum_{i=1}^r m_i$. Since every S^i_μ for a node $\mu \in K(B)$ has one leaf per child of μ, by Lemma 7 their intersection S_μ can be computed in $O(r \cdot |\deg(\mu)|)$ time. Hence in total we need $\sum_{\mu \in K(B)} O(r \cdot |\deg(\mu)|) = O(n)$ time to compute S. For every i, by Theorem 2, we can compute φ_i in $O(n_i + m_i)$ time. By Lemma 7 we can also find out in $O(n)$ time which Q-nodes are merged and in which direction and hence the construction of φ in total takes $O(n + m)$ time. □

Now consider the case where $(B, S^1, \varphi_1), \ldots, (B, S^r, \varphi_r)$ are the constrained modular decompositions we get from $T_1|_H, \ldots, T_r|_H$. By Lemma 11 and Theorem 2 we have $\text{to}(B, S, \varphi) = \bigcap_{i=1}^r \text{to}(B, S_i, \varphi_i) = \bigcap_{i=1}^r \text{to}(T_i, D)$ and by Lemma 3 $G_1, \ldots, G_r$ are simultaneous comparability graphs if and only if φ is satisfiable and S does not contain the null tree.

Theorem 3 SimOrient* *can be solved in linear time.*

Proof Let $G_1 = (V_1, E_1), \ldots, G_r = (V_r, E_r)$ be r-sunflower graphs with $n_i = |V_i|$ and $m_i = |E_i|$ for all $1 \leq i \leq r$ and let $n = \sum_{i=1}^r n_i$, $m = \sum_{i=1}^r m_i$. We solve SimOrient* as follows.

1. Compute the canonical modular decomposition T_i for every G_i and the canonical modular decomposition B of H in $O(n+m)$ time by McConnell and Spinrad [27].
2. Compute $T_i|_H$ for every i in $O(n)$ time in total.
3. Compute (B, S^i, φ_i) for every i, in $O(n+m)$ time in total by Theorem 2.
4. Compute (B, S, φ) in linear time by Lemma 11.
5. Check whether S contains the null tree and whether φ is satisfiable in linear time.

We execute Step 5 as follows. For $i \in \{1, \ldots, r\}$, φ_i contains one variable and one constraint per prime node in B, one variable per prime node in $T_i|_H$ and one variable and one constraint per Q-node in S^i_μ. Since S^i_μ has $O(n_i)$ nodes, it contains $O(n_i)$ Q-nodes. Hence in total every φ_i contains $O(n_i)$ variables and clauses. Note that φ' contains one clause per Q-node in the PQ-trees in S. Hence φ' contains $O(n)$ clauses and variables and thus the 2-SAT formula φ can be solved in $O(n)$ time by Aspvall et al. [34].

If S does not contain the null tree and φ has a solution, we get simultaneous transitive orientations of $G_1, \ldots, G_r$ in linear time by proceeding as follows. We orient every complete quotient graph of B according to a total order induced by the corresponding PQ-tree where every Q-node is oriented according to the solution of φ. For a prime quotient graph $H[\mu]$ we choose D_μ if in the chosen solution of φ we have $x_\mu = \texttt{true}$ and D_μ^{-1} otherwise. Together, all these orientations of quotient graphs of B induce a transitive orientation on H and by applying the linear-time algorithm from Sect. 3 to solve OrientExt we can extend it to a transitive orientation of G_i for every $i \in \{1, \ldots, r\}$. □

5 Permutation Graphs

We give algorithms that solve RepExt(Perm) and SimRep*(Perm) in linear time using modular decomposition and the results from Sects. 3 and 4. To do so, we need the following definitions and observations. Let $G = (V, E)$ be a permutation graph and let D be a representation of G. We denote the upper horizontal line of D by L_1 and the lower line by L_2.

Proposition 4 ([35]) *Let $G = (V, E)$ be a permutation graph and let T be the canonical modular decomposition of G. Then for every representation D of G and for every $\mu \in T$, the vertices in $L(\mu)$ appear consecutively along both L_1 and L_2.*

Let G be a permutation graph and let T be the canonical modular decomposition of G. We use Proposition 4 to show that for node $\mu \in T$ we can compute a permutation diagram of $G[L(\mu)]$ by iteratively replacing a line segment representing a descendent ν of μ in T by a permutation diagram of $G[\nu]$.

Theorem 5 *Let G be a permutation graph and let T be the canonical modular decomposition of G. There is a bijection ϕ between the permutation diagrams of G and the choice of a permutation diagram D_μ for each quotient graph $\mu \in T$. Both ϕ and ϕ^{-1} can be computed in $O(n)$ time.*

Proof To compute a permutation diagram of G from $\{D_\mu | \mu \in T\}$ we traverse T bottom-up and compute for every $\mu \in T$ a permutation diagram representing $G[L(\mu)]$

as follows. For every child ν of μ replace the line segment representing ν in D_μ by the permutation diagram of $G[L(\nu)]$. For every permutation diagram we store two doubly-linked lists containing the order of labels along the two horizontal lines, respectively. Then the replacements described above take linear time in total.

Conversely, assume that a permutation diagram D for G is given as two double linked lists $l_1(D)$ and $l_2(D)$ containing the order of labels along the two horizontal lines, respectively. We traverse T bottom-up and consider all leaves as initially visited. Let l_1 and l_2 be two double linked lists and initially $l_1 = l_1(D)$ and $l_2 = l_2(D)$. When visiting a non-leaf node $\mu \in T$ we compute the two double linked lists $l_1(\mu)$ and $l_2(\mu)$ representing its permutation diagram as follows. As an invariant we claim that at any time l_1 and l_2 are sublists of $l_1(D)$ and $l_2(D)$ and represent the permutation diagram of a subgraph of G. Further l_1 and l_2 contain for every visited child ν of an unvisited node exactly one entry corresponding to a leaf in $L(\nu)$. Clearly the invariant holds at the beginning of the traversal. Now assume we visit a non-leaf node μ and the invariant holds. Let $l_1(\mu)$ and $l_2(\mu)$ be the sublist of l_1 and l_2 consisting of all entries corresponding to leaves in $L(\mu)$, respectively. The invariant together with Proposition 4 gives us that when visiting a node μ, $l_1(\mu)$ and $l_2(\mu)$ are consecutive sublists of l_1 and l_2. To compute $l_1(\mu)$ and $l_2(\mu)$ we start at an arbitrary child ν of μ in l_1 and l_2 and search for the right and left end of $l_1(\mu)$ and $l_2(\mu)$, i.e. on both sides we search for the first entry that does not correspond to a child of μ. Finally we remove all list entries corresponding to a leaf in $L(\mu)$ that is not in $L(\nu)$ in l_1 and l_2. After this step the invariant still holds. By definition of the quotient graphs and since they contain exactly one representative per child of μ, $l_1(\mu)$ and $l_2(\mu)$ represent the permutation diagram for $G[\mu]$. In total these computations take linear time since for every $\mu \in T$ the number of visited list entries is in $O(\deg(\mu))$. □

5.1 Extending Partial Representations

For solving RepExt(Perm) efficiently, we exploit that a given partial representation D' of a permutation graph G is extendible if and only if, for every prime node μ in the canonical modular decomposition of G, the partial representation of $G[\mu]$ induced by D' is extendible. Since $G[\mu]$ has only a constant number of representations this can be checked in linear time.

In the following, let $G = (V, E)$ be a permutation graph and let D' be a corresponding partial representation of a subgraph $H = (V', E')$ of G. Furthermore, let T be the canonical modular decomposition for G and let μ be a node in T. Let $U'_\mu \subseteq V'$ be a (not necessarily maximal) μ-set containing as many vertices in V' as possible. Let D'_μ be the partial representation of $G[\mu]$ we get from D' by removing all line segments corresponding to vertices not in U'_μ and replacing every label u by the label $\text{rep}_\mu(u)$. Let U_μ be a maximal μ-set with $U'_\mu \subseteq U_\mu$.

Lemma 13 *Let D' be a partial representation of G. Then D' can be extended to a representation D of G if and only if for every inner node μ in T, D'_μ can be extended to a representation D_μ of $G[\mu]$.*

Proof Assume that D' can be extended to a representation D of G. Then for every inner node μ in T, D restricted to U'_μ where every label u is replaced by the label $\text{rep}_\mu(u)$ is an extension of D'_μ representing $G[\mu]$.

Conversely assume that for every inner node μ in T, D'_μ can be extended to a representation D_μ of $G[\mu]$. By Theorem 5 we get a permutation diagram D representing G from the D_μs. We show that D extends D'.

Let L_1 and L_2 denote the upper and the bottom line of D, respectively. Now for every pair $u, v \in V'$ let $\mu = \text{lca}_T(u, v)$. By Proposition 4 $L(\mu)$ appears consecutively along L_1 and L_2. Hence independently of the choice of U'_μ, in D_μ $\text{rep}(u)$ and $\text{rep}(v)$ appear in the same order as the labels corresponding to u and v in D' along both horizontal lines. Thus D restricted to V' coincides with D'. □

Theorem 6 RepExt(Perm) *can be solved in linear time.*

Proof Let $G = (V, E)$ be a permutation graph with n vertices and m edges and let D' be a permutation diagram of an induced subgraph $H = (V', E')$ of G. We compute the canonical modular decomposition T of G in $O(n + m)$ time [27]. By Lemma 13, it suffices to check whether D'_μ can be extended to a representation of $G[\mu]$ for every $\mu \in T$.

If $G[\mu]$ is empty or complete, we can easily extend D'_μ to a representation D_μ of $G[\mu]$. If μ is prime, each of $G[\mu]$ and $\overline{G}[\mu]$ has exactly two transitive orientations where one is the reverse of the other. Hence there exist only four permutation diagrams representing $G[\mu]$ [24]. Note that given an arbitrary permutation diagram D_μ for $G[\mu]$, we get the other three representations by either reversing the order along both horizontal lines, switching L_1 and L_2, or applying both. Hence we can compute all four representations in linear time and check whether one of them contains D'_μ.

In the positive case, by Theorem 5 we get a representation D of G that extends D' from the representations of the quotient graphs in linear time. □

5.2 Simultaneous Representations

Recall that a graph G is a permutation graph if and only if G is a comparability and a co-comparability graph. To solve SimRep*(Perm) efficiently, we build on the following characterization due to Jampani and Lubiw.

Proposition 7 ([19]) *Sunflower permutation graphs are simultaneous permutation graphs if and only if they are simultaneous comparability graphs and simultaneous co-comparability graphs.*

Let $G_1, \ldots, G_r$ be sunflower permutation graphs. It follows from Proposition 7 that we can solve SimRep*(Perm) by solving SimOrient* (see Sect. 4) for $G_1, \ldots, G_r$ and their complements $\overline{G_1}, \ldots, \overline{G_r}$. Since the algorithm from Sect. 4 needs time linear in the size of the input graphs and $\overline{G}$ may have a quadratic number of edges, this approach takes quadratic time in total. This already improves the cubic runtime of the algorithm presented by Jampani and Lubiw [19].

We show that it is possible to use the algorithm from Sect. 4 to solve also SimOrient* for co-comparability graphs and thus SimRep*(Perm) in linear time.

Let G be a permutation graph with induced subgraph H. Recall that a graph G and its complement have the same canonical modular decomposition, they only differ in the type of the nodes. For a node μ with $G[\mu]$ prime also $\overline{G}[\mu]$ is prime; if $G[\mu]$ is complete or empty, $\overline{G}[\mu]$ is empty or complete, respectively [26].

Let B be the canonical modular decomposition of H and let T be the canonical modular decomposition of G restricted to H. The goal is to compute constrained modular decompositions $(B, S^1, \varphi_1), \ldots, (B, S^r, \varphi_r)$ such that for every $i \in \{1, \ldots, r\}$ (B, S^i, φ_i) contains exactly those transitive orientations of $\overline{H}$ that can be extended to a transitive orientation of $\overline{G}_i$, and (B, S, φ) with $\text{to}(B, S, \varphi) = \bigcap_{i=1}^{r} \text{to}(B, S^i, \varphi_i)$. In linear time we cannot explicitly compute $\overline{G}$ and the corresponding quotient graphs. Hence we cannot store a default orientation for the prime quotient graphs $\overline{H}[\mu]$. We can, however, compute a *default permutation diagram* D_μ representing $\overline{H}[\mu]$ from its default orientation O_μ in linear time [27]. We apply the algorithm from Sect. 4. The 2-SAT-formula ψ describing the dependencies between the orientations of the prime nodes can be computed in $O(|V(H)| + |E(H)|)$ time since when it is required to check whether an oriented edge uv is contained in the default orientation of a prime quotient graph $\overline{H}[\mu]$, we do this in constant time by checking for the non-edge uv in $H[L(\mu)]$ whether $\text{rep}_\mu(u) <_\mu \text{rep}_\mu(v)$. We can determine one non-edge for each prime quotient graph of B in $O(|E(H)|)$ time in total. Note that one non-edge per prime quotient graph suffices to answer the queries. Further we have to compute the PQ-trees for the complete nodes in T and χ. Note that the only step in the computation of the PQ-trees that takes potentially quadratic time for $\overline{H}$ is the computation of the order of the children for the Q-nodes. But since every prime node is labeled with a default representation, we can compute the PQ-trees and χ in $O(|V(H)|)$ time instead of $O(|V(\overline{H})| + |E(\overline{H})|)$ time, since we get the order of the children of a Q-node directly from the corresponding permutation diagram.

Theorem 8 SIMREP*(PERM) *can be solved in linear time.*

Proof Let φ be the 2-SAT formula we get for H and let $\overline{\varphi}$ be the formula from $\overline{H}$. Further let S and $\overline{S}$ be the set of intersected PQ-trees we get for the input graphs $G_1, \ldots, G_r$ and $\overline{G_1}, \ldots, \overline{G_r}$, respectively. Then $G_1, \ldots, G_r$ are simultaneous permutation graphs if and only if φ and $\overline{\varphi}$ are satisfiable and neither S nor $\overline{S}$ contain the null tree. This can be checked in linear time. In the positive case we proceed as follows.

1. For every quotient graph in B compute the permutation diagram induced by the solutions of φ and $\overline{\varphi}$.
2. Compute a permutation diagram D_H representing H from the representations of the quotient graphs (see Theorem 5).
3. For every input graph G_i use the algorithm from Sect. 5.1 to extend D_H to a representation of G_i.

More precisely, Step 2 works as follows. For complete or empty nodes μ we construct the representation by choosing a linear order of their children represented by the corresponding PQ-tree where every Q-node is oriented according to a solution of φ or $\overline{\varphi}$, for the upper line. If $H[\mu]$ is empty, we choose the same order for the bottom line, if $H[\mu]$ is complete, the bottom line is labeled with the reversed order.

For a prime quotient graph μ we distinguish four cases. Let x_μ be the variable encoding the orientation of $H[\mu]$ and let y_μ be the variable encoding the orientation of $\overline{H}[\mu]$. If $x_\mu = \texttt{true}, y_\mu = \texttt{true}$ we choose the default representation D_μ, if $x_\mu = \texttt{true}, y_\mu = \texttt{false}$ we reverse the orders along both horizontal lines of D_μ, if $x_\mu = \texttt{false}, y_\mu = \texttt{true}$ we switch the orders along the horizontal lines of D_μ and if $x_\mu = \texttt{false}, y_\mu = \texttt{false}$ we reverse the orders along both horizontal lines of D_μ and switch them. □

6 Circular Permutation Graphs

In this section we give efficient algorithms for solving RepExt (CPerm) and SimRep*(CPerm) based on the linear time algorithms for solving RepExt (Perm) and SimRep*(Perm) and the *switch* operation.

Let $G = (V, E)$ be a circular permutation graph. *Switching* a vertex v in G, i.e., connecting it to all vertices it was not adjacent to in G and removing all edges to its former neighbors, gives us the graph $G_v = (V, E_v)$ with $E_v = \binom{V}{2} \backslash E$. The graph we obtain by switching all neighbors of a vertex v we denote by $G_{N(v)}$.

Let C be a circular permutation diagram that represents a permutation graph $G = (V, E)$ and let $v \in V$ be a vertex in G. The chord v in C can be *switched* to the chord v', if v' has the same endpoints as v, but intersects exactly the chords v does not intersect in C. Hence, this modified circular permutation diagram C' is a representation of G_v [36]. Here it suffices to know that there exists a v' that v can be switched to, hence, we use the term *switching chord* v, without further specifying v'.

6.1 Extending Partial Representations

In this section we show that RepExt(CPerm) can be solved via a linear-time Turing reduction to RepExt(Perm) using the switch operation.

Let G be a circular permutation graph and let C_p be a circular permutation diagram representing an induced subgraph H of G. Let v be an arbitrary vertex in G and let $G' = G_{N(v)}$. The circular permutation diagram we obtain by switching all chords corresponding to neighbors of v in C_p we denote by C'_p. Since G' is a permutation graph C'_p can be transformed into a permutation diagram D'_p representing the same induced subgraph. Observe that C_p is extendible to a representation of G if and only if D'_p can be extended to a representation of G' since an extension of D'_p can easily be transformed back into a circular permutation diagram extending C_p. This can be checked in linear time using the algorithm from Sect. 5.1.

To achieve a linear runtime in total we have to switch the neighborhood of a vertex v with minimum degree in G. In case $v \in V(H)$ we can easily transform C'_p into a permutation diagram D'_p representing the same induced subgraph by opening C'_p along the isolated chord v. If $v \notin V(H)$ it is more difficult to find a position where we can open C'_p.

Theorem 9 RepExt(CPerm) *can be solved in linear time.*

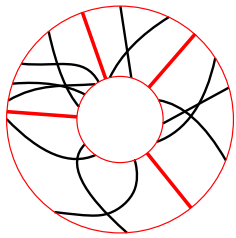

Fig. 7 The circular permutation representation C of a graph with four connected components. The red line segments mark the positions where C can be opened (Color figure online)

Proof Let G be a circular permutation graph and let C' be a circular permutation diagram representing an induced subgraph H of G. Let v be a vertex of minimum degree in G. Then we compute G' in linear time [37]. In case $v \in V(H)$ we open C'_p along the isolated chord v and check in linear time whether the resulting permutation diagram can be extended to a representation of G'. If $v \notin V(H)$ we proceed as follows.

Let H' be the graph we obtain from H by switching all vertices that are adjacent to v in G and let $CC(H')$ be the set of all connected components in H'. Note that in C'_p the endpoint of chords corresponding to vertices of the same connected component of H' appear as consecutive blocks along both the outer and the inner circle and that the only positions where we can open C'_p are *between* these blocks; see Fig. 7.

If $|CC(H')| = 1$ or $|CC(H')| = 2$ there are only one or two positions, respectively, where we can open C'_p. Hence in these cases we can construct all one or two possible permutation diagrams D'_p representing H' and check whether one of them is extendible. Else we distinguish several further cases.

Case 1: $|CC(H')| > 2$ and there exists a vertex u in G such that u has a neighbor in every connected component of H' but is not adjacent to all vertices in H. Note that if more than two connected components in $CC(H')$ contain vertices not adjacent to u, C'_p is not extendible. The same holds if two connected components in $CC(H')$ whose corresponding blocks are not adjacent in C'_p, contain vertices not adjacent to u. Else we distinguish further subcases.

Case 1a: Two connected components in $CC(H')$ contain vertices not adjacent to u and the corresponding blocks are adjacent in C'_p. Then the only position where we can open C'_p is between these two blocks.

Case 1b: Only one connected components in $CC(H')$ contains vertices not adjacent to u. Then we can either open to the left or to the right of the block. In this case we check whether one of the two possibilities gives us an extendible permutation diagram.

Case 2: There exists no vertex u in G that has a neighbor in every connected component of H' but is not adjacent to every vertex in H. Then we remove all vertices from G that are adjacent to every other vertex in H. If G is empty afterwards, we can choose an arbitrary gap between two adjacent blocks and open C'_p there. Else we iteratively merge for each remaining vertex u all connected components of H' that contain a vertex adjacent to u. Since G is connected we end up with either one or two blocks which gives us only one or two positions to open C'_p, respectively. We check whether one of them leads to an extendible permutation diagram. □

6.2 The Simultaneous Representation Problem

In this section we show that SimRep*(CPerm) can be solved via a quadratic-time Turing reduction to SimRep*(Perm) using the switch operation. Here we need to switch the neighborhood of a vertex v shared by all input graphs. Hence in contrast to the reduction in Section 6.1, where we switched the neighborhood of a vertex of minimum degree, the graph $G_{N(v)}$ may have quadratic size.

Lemma 14 *Let $G_1, G_2, \ldots, G_r$ be sunflower circular permutation graphs sharing an induced subgraph H. Let v be a vertex in H and let G_i' be the graph we obtain by switching all neighbors of vertex v in G_i for $i \in \{1, \ldots, r\}$. Then $G_1, G_2, \ldots, G_r$ are simultaneous circular permutation graphs if and only if $G_1', G_2', \ldots, G_r'$ are simultaneous permutation graphs.*

Proof Recall that $G_1', G_2', \ldots, G_r'$ are indeed permutation graphs [36]. Now let $G_1, G_2, \ldots, G_r$ be simultaneous circular permutation graphs. Then there exist circular permutation diagrams $C_1, C_2, \ldots, C_r$ such that for every $1 \leq i \leq r$, C_i represents G_i and all C_is coincide restricted to the vertices of H. We get a circular permutation diagram C_i' representing G_i' by switching all chords corresponding to neighbors of vertex v in C_i. Since the order of the vertices along the inner and outer circle is not affected by the switch operation, we know that all C_i's coincide restricted to the vertices of H. Recall that after switching all neighbors of chord v in a circular permutation diagram, no chord intersects v any more and hence we obtain a permutation diagram D_i' representing G_i' by opening C_i' along the chord v. Then the D_i's also coincide on the vertices of H and hence $G_1', G_2', \ldots, G_r'$ are simultaneous permutation graphs.

Conversely let $G_1', G_2', \ldots, G_r'$ be simultaneous permutation graphs. Then there exist permutation diagrams $D_1', D_2', \ldots, D_r'$ such that for every $1 \in \{1, \ldots, r\}$, D_i' represents G_i' and all D_i's coincide on the vertices of H. Recall that we can transform every linear permutation diagram D_i' into a circular permutation diagram C_i' representing G_i'. Note that the C_i's also coincide on the vertices of H. Now we obtain a circular permutation diagram C_i representing G_i by switching all chords that correspond to vertices adjacent to v. The C_is still coincide on the vertices of H since the switch operation does not change the order of the vertices along the outer or the inner circle of a circular permutation diagram and chords corresponding to vertices in H are either switched in every C_i or in none of them. □

Theorem 10 SimRep*(CPerm) *can be solved in $O(n^2)$ time.*

Proof Let $G_1, \ldots, G_r$ be sunflower circular permutation graphs. Let v be a vertex in H and let G_i' be the graph we obtain by switching all neighbors of vertex v in G_i for $i \in \{1, \ldots, r\}$. By Lemma 14 to solve SimRep(CPerm) for $G_1, G_2, \ldots, G_r$ it suffices to solve SimRep(Perm) for $G_1', G_2', \ldots, G_r'$. Computing $G_1', G_2', \ldots, G_r'$ takes quadratic time. In Sect. 5.2 we have seen that SimRep(Perm) can be solved in linear time for the sunflower permutation graphs $G_1', \ldots, G_r'$, hence in total we need quadratic time to solve the problem SimRep*(CPerm).

If $G_1, G_2, \ldots, G_r$ are simultaneous circular permutation graphs we get corresponding simultaneous representations by transforming simultaneous linear permutation

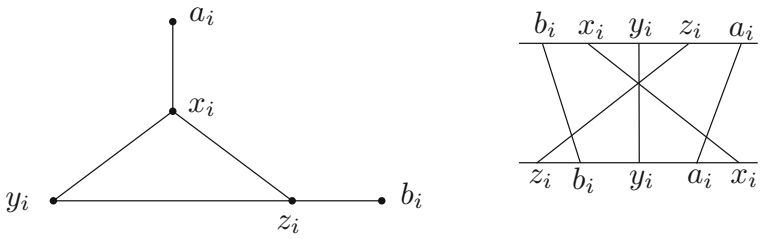

Fig. 8 The (permutation) graph G_i corresponding to the $i-th$ triple (x_i, y_i, z_i) of a TOTALORDERING instance with corresponding permutation diagram

diagrams representing $G'_1, G'_2, \ldots, G'_r$ into circular permutation diagrams and switching all chords corresponding to a neighbor of vertex v in H. This takes quadratic time in total. □

7 Simultaneous Orientations and Representations for General Comparability Permutation Graphs

We show that the simultaneous orientation problem for comparability graphs and the simultaneous representation problem for permutation graphs are NP-complete in the non-sunflower case.

Theorem 11 SIMORIENT *for k comparability graphs where k is part of the input is NP-complete.*

Proof Clearly the problem is in NP.

To show the NP-hardness, we give a reduction from the known NP-complete problem TOTALORDERING [38], which is defined as follows. Given a finite set S and a finite set T of triples of elements in S, decide whether there exists a total ordering $<$ of S such that for all triples $(x, y, z) \in T$ either $x < y < z$ or $x > y > z$. Let H_T be an instance of TOTALORDERING with $s = |S|$ and $t = |T|$. We number the triples in T with $1, \ldots, t$ and denote the ith triple by (x_i, y_i, z_i). We construct an instance H_S of SIMORIENT, consisting of undirected graphs $G_0, \ldots, G_t$ as follows.

- $G_0 := (S, E_0)$ is the complete graph with one vertex for each element in S.
- $G_i := (V_i, E_i)$ for $1 \leq i \leq t$ is the graph with vertex set $V_i = \{x_i, y_i, z_i, a_i, b_i\}$, where $a_i, b_i \notin S$ are new vertices, and edges $E_i = \{x_i a_i, x_i y_i, x_i z_i, y_i z_i, z_i b_i\}$; see Fig. 8.

For every $1 \leq i \leq t$, the graph G_i has exactly two transitive orientations, namely $\{\overrightarrow{x_i a_i}, \overrightarrow{x_i y_i}, \overrightarrow{x_i z_i}, \overrightarrow{y_i z_i}, \overrightarrow{b_i z_i}\}$ and its reversal. We now have to show that H_T has a solution if and only if $G_0, \ldots, G_t$ are simultaneous comparability graphs.

First, assume that $G_0, \ldots, G_t$ are simultaneous comparability graphs. Hence there exist orientations $T_0, \ldots, T_t$ such that for every $0 \leq i \leq t$, T_i is an orientation of G_i and for every $j, k \in \{0, \ldots t\}$ every edge in $E_j \cap E_k$ is oriented in the same way in both T_j and T_k. Then the orientation of the complete graph G_0 implies a total order on the elements of T, where $u < v$ if and only if the edge uv is oriented from u to v.

By construction, there are only two valid transitive orientations for G_i. The one given above implies $x_i < y_i < z_i$ and for the reverse orientation we get $z_i < y_i < x_i$. Hence the received total order satisfies that for every triple $(x, y, z) \in T$ we have either $x < y < z$ or $x > y > z$.

Conversely, assume that H_T has a solution. Then there exists a total order $<$ such that for all triples $(x, y, z) \in T$ either $x < y < z$ or $x > y > z$ holds. We get a transitive orientation T_i of G_i for $1 \leq i \leq t$ by orienting all edges between the vertices x_i, y_i and z_i towards the greater element according to the order $<$. If x_i is the smallest element of the triple, then we choose $\overrightarrow{x_i a_i}$ and $\overrightarrow{b_i z_i}$, else z_i is the smallest element and we choose $\overrightarrow{a_i x_i}$ and $\overrightarrow{z_i b_i}$. Finally, we orient the edges in G_0 also towards the greater element according to $<$. This gives us orientations $T_0, \ldots, T_t$ of $G_0, \ldots, G_t$ with the property that for every $j, k \in \{0, \ldots t\}$ every edge in $E_j \cap E_k$ is oriented in the same way in both T_j and T_k. Hence $G_0, \ldots, G_t$ are simultaneous comparability graphs.

Hence the instance H_T of TOTALORDERING has a solution if and only if the instance H_S is a yes-instance of SIMORIENT. Since H_S can be constructed from H_T in polynomial time, it follows that SIMORIENT is NP-complete. □

Theorem 12 SIMREP(PERM) *for k permutation graphs where k is not fixed is NP-complete.*

Proof Clearly the problem is in NP.

To show the NP-hardness, we use the same reduction as in the proof of Theorem 11. Let H_T be the corresponding instance of TOTALORDERING. Note that $G_0, \ldots, G_t$ are permutation graphs and thus $\overline{G_0}, \ldots, \overline{G_t}$ are comparability graphs.. We already know that H_T has a solution if and only if $G_0, \ldots, G_t$ are simultaneous comparability graphs. Hence it remains to show that $G_0, \ldots, G_t$ are simultaneous co-comparability graphs if H_T has a solution. Note that $E(\overline{G_0})$ is empty and for every $1 \leq i \leq t$ for every edge $uv \in E(\overline{G_i})$ at least one of the endpoints u and v is in $\{a_i, b_i\}$. Hence, $\overline{G_0}, \ldots, \overline{G_t}$ pairwise do not share any edges and thus they are also simultaneous comparability graphs. □

8 Conclusion

We showed that the orientation extension problem and the simultaneous orientation problem for sunflower comparability graphs can be solved in linear time using the concept of modular decompositions. Further we were able to use these algorithms to solve the partial representation problem for permutation and circular permutation graphs and the simultaneous representation problem for sunflower permutation graphs also in linear time. For the simultaneous representation problem for circular permutation graphs we gave a quadratic-time algorithm. The non-sunflower case of the simultaneous orientation problem and the simultaneous representation problem turned out to be NP-complete.

It remains an open problem whether the simultaneous representation problem for sunflower circular permutation graphs can be solved in subquadratic time. Furthermore

it would be interesting to examine whether the concept of modular decomposition is also applicable to solve the partial representation and the simultaneous representation problem for further graph classes, e.g. trapezoid graphs. There may also be other related problems that can be solved for comparability, permutation and circular permutation graphs with the concept of modular decompositions.

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

Funding Open Access funding enabled and organized by Projekt DEAL. This work was supported by grant RU 1903/3-1 of the German Research Foundation (DFG).

Declarations

Conflict of interest The authors have no known conflicts of interest to declare.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T., Vyskočil, T.: Extending partial representations of interval graphs. Algorithmica **78**(3), 945–967 (2017). https://doi.org/10.1007/s00453-016-0186-z
2. Klavík, P., Kratochvíl, J., Otachi, Y., Rutter, I., Saitoh, T., Saumell, M., Vyskočil, T.: Extending partial representations of proper and unit interval graphs. Algorithmica **77**(4), 1071–1104 (2017). https://doi.org/10.1007/s00453-016-0133-z
3. Klavík, P., Kratochvíl, J., Krawczyk, T., Walczak, B.: Extending partial representations of function graphs and permutation graphs. In: Epstein, L., Ferragina, P. (eds.) 20th Annual European Symposium on Algorithms (ESA'12). Lecture Notes in Computer Science, pp. 671–682. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33090-2_58
4. Chaplick, S., Fulek, R., Klavík, P.: Extending partial representations of circle graphs. J. Graph Theory **91**(4), 365–394 (2019). https://doi.org/10.1002/jgt.22436
5. Klavík, P., Kratochvíl, J., Otachi, Y., Saitoh, T.: Extending partial representations of subclasses of chordal graphs. Theoret. Comput. Sci. **576**, 85–101 (2015). https://doi.org/10.1016/j.tcs.2015.02.007
6. Krawczyk, T., Walczak, B.: Extending partial representations of trapezoid graphs. In: Bodlaender, H.L., Woeginger, G.J. (eds.) 43rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'17), pp. 358–371. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-68705-6_27
7. Angelini, P., Battista, G.D., Frati, F., Jelínek, V., Kratochvíl, J., Patrignani, M., Rutter, I.: Testing planarity of partially embedded graphs. ACM Trans. Algorithms **11**(4), 1–42 (2015). https://doi.org/10.1145/2629341
8. Jelínek, V., Kratochvíl, J., Rutter, I.: A Kuratowski-type theorem for planarity of partially embedded graphs. Comput. Geom. **46**(4), 466–492 (2013). https://doi.org/10.1016/j.comgeo.2012.07.005
9. 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
10. 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. 1–19. Schloss Dagstuhl

- Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2020). https://doi.org/10.4230/LIPIcs.ICALP.2020.43
11. Chaplick, S., Dorbec, P., Kratochvíl, J., Montassier, M., Stacho, J.: Contact representations of planar graphs: Extending a partial representation is hard. In: Kratsch, D., Todinca, I. (eds.) 40th International Workshop on Graph-theoretic Concepts in Computer Science (WG'14). Lecture Notes in Computer Science, vol. 8747, pp. 139–151. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-319-12340-0_12
12. Chaplick, S., Kindermann, P., Klawitter, J., Rutter, I., Wolff, A.: Extending partial representations of rectangular duals with given contact orientations. In: Calamoneri, T., Corò, F. (eds.) Proceedings of the 12th International Conference on Algorithms and Complexity, (CIAC '21) Lecture Notes in Computer Science, vol. 12701, pp. 340–353. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-75242-2_24
13. Bläsius, T., Kobourov, S.G., Rutter, I.: Simultaneous embedding of planar graphs. In: Tamassia, R. (ed.) Handbook of Graph Drawing and Visualization, pp. 349–381. CRC Press, Boca Raton (2013)
14. Brass, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D.P., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. Comput. Geom. **36**(2), 117–130 (2007). https://doi.org/10.1016/j.comgeo.2006.05.006
15. Gassner, E., Jünger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous graph embeddings with fixed edges. In: Fomin, F.V. (ed.) 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG'06), pp. 325–335. Springer, Heidelberg (2006). https://doi.org/10.1007/11917496_29
16. Schaefer, M.: Toward a theory of planarity: Hanani-Tutte and planarity variants. In: 20th International Symposium on Graph Drawing (GD'12), pp. 162–173. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-36763-2_15
17. Angelini, P., Da Lozzo, G., Neuwirth, D.: On some $\mathcal{NP}$-complete SEFE problems. In: Pal, S.P., Sadakane, K. (eds.) roceedings of the 8th International Workshop on Algorithms and Computation (WALCOM'14). Lecture Notes in Computer Science, vol. 8344, pp. 200–212. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-319-04657-0_20
18. Estrella-Balderrama, A., Gassner, E., Jünger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous geometric graph embeddings. In: Hong, S., Nishizeki, T., Quan, W. (eds.) Proceedings of 15th International Symposium on Graph Drawing (GD'07), pp. 280–290. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77537-9_28
19. Jampani, K.R., Lubiw, A.: The simultaneous representation problem for chordal, comparability and permutation graphs. J. Graph Algorithms Appl. **16**(2), 283–315 (2012). https://doi.org/10.7155/jgaa.00259
20. Jampani, K.R., Lubiw, A.: Simultaneous interval graphs. In: Cheong, O., Chwa, K., Park, K. (eds.) Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC'10), pp. 206–217. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17517-6_20
21. Bläsius, T., Rutter, I.: Simultaneous PQ-ordering with applications to constrained embedding problems. ACM Trans. Algorithms **12**(2), 1–46 (2015). https://doi.org/10.1145/2738054
22. Bok, J., Jedličková, N.: A note on simultaneous representation problem for interval and circular-arc graphs. Computing Research Repository (2018)
23. Rutter, I., Strash, D., Stumpf, P., Vollmer, M.: Simultaneous representation of proper and unit interval graphs. In: Bender, M.A., Svensson, O., Herman, G. (eds.) 27th Annual European Symposium on Algorithms (ESA'19), vol. 144, p. 80. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2019)
24. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Elsevier, London (2004)
25. Gilmore, P.C., Hoffman, A.J.: A characterization of comparability graphs and of interval graphs. Can. J. Math. **16**, 539–548 (1964)
26. Gallai, T.: Transitiv orientierbare graphen. Acta Math. Acad. Sci. Hung. **18**(1–2), 25–66 (1967)
27. McConnell, R.M., Spinrad, J.P.: Modular decomposition and transitive orientation. Discrete Math. **201**(1–3), 189–241 (1999). https://doi.org/10.1016/S0012-365X(98)00319-7
28. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984). https://doi.org/10.1137/0213024
29. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT press, Cambridge (2009)

30. Gabow, H.N., Tarjan, R.E.: A linear-time algorithm for a special case of disjoint set union. J. Comput. Syst. Sci. **30**(2), 209–221 (1985). https://doi.org/10.1016/0022-0000(85)90014-5
31. Booth, K.S.: PQ-tree algorithms. PhD thesis, University of California, Berkeley (1975)
32. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comput. Syst. Sci. **13**(3), 335–379 (1976). https://doi.org/10.1016/S0022-0000(76)80045-1
33. Haeupler, B., Jampani, K.R., Lubiw, A.: Testing simultaneous planarity when the common graph is 2-connected. J. Graph Algorithms Appl. **17**(3), 147–171 (2013). https://doi.org/10.7155/jgaa.00289
34. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified Boolean formulas. Inf. Process. Lett. **8**(3), 121–123 (1979). https://doi.org/10.1016/0020-0190(79)90002-4
35. Montgolfier, F.d.: Décomposition modulaire des graphes: théorie, extensions et algorithmes. PhD thesis, Montpellier 2 University (2003)
36. Rotem, D., Urrutia, J.: Circular permutation graphs. Networks **12**(4), 429–437 (1982). https://doi.org/10.1002/net.3230120407
37. Sritharan, R.: A linear time algorithm to recognize circular permutation graphs. Netw. Int. J. **27**(3), 171–174 (1996). https://doi.org/10.1002/(SICI)1097-0037(199605)27:3<171::AID-NET1>3.0.CO;2-F
38. Opatrny, J.: Total ordering problem. SIAM J. Comput. **8**(1), 111–114 (1979). https://doi.org/10.1137/0208008

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.