Simultaneous Representation of Interval Graphs in the Sunflower Case

Ignaz Rutter
Faculty of Computer Science and Mathematics, University of Passau, Germany

Peter Stumpf
Faculty of Computer Science and Mathematics, University of Passau, Germany

Abstract
A simultaneous representation of (vertex-labeled) graphs $G_1, \ldots, G_k$ consists of a (geometric) intersection representation $R_i$ for each graph $G_i$ such that each vertex $v$ is represented by the same geometric object in each $R_i$ for which $G_i$ contains $v$. While Jampani and Lubiw showed that the existence of simultaneous interval representations for $k = 2$ can be tested efficiently (2010), testing it for graphs where $k$ is part of the input is NP-complete (Bok and Jedličková, 2018). An important special case of simultaneous representations is the sunflower case, where $G_i \cap G_j = (V(G_i) \cap V(G_j), E(G_i) \cap E(G_j))$ is the same graph for each $i \neq j$. We give an $O\left(\sum_{i=1}^{k} (|V(G_i)| + |E(G_i)|)\right)$-time algorithm for deciding the existence of a simultaneous interval representation for the sunflower case, even when $k$ is part of the input. This answers an open question of Jampani and Lubiw.

1 Introduction
For a family of geometric objects, the intersection graph is a graph that has for each object a vertex such that two vertices are adjacent if and only if their objects intersect. Its representation is the assignment of the objects to the vertices. In this paper, we consider interval representations, which are assignments of intervals on the real line to the vertices of a graph $G$ such that two vertices of $G$ are adjacent if and only if their intervals intersect. Graph $G$ is an interval graph if it has such a representation; see Figure 1.

A fundamental problem in the area of intersection graphs is the recognition problem, where the task is to decide whether a given graph $G$ admits a particular type of (geometric) intersection representation. The simultaneous representation problem is a generalization of the recognition problem that asks for $k$ input graphs $G_1, \ldots, G_k$ (with vertex labels) whether there exist corresponding representations $R_1, \ldots, R_k$ such that each vertex $v$ that is shared by two graphs $G_i$ and $G_j$ is represented by the same geometric object in $R_i$ and in $R_j$. For ease of notation, we refer to $G = (G_1, \ldots, G_k)$ as a simultaneous graph, and

Figure 1 An interval graph $G$, and an interval representation $R$ of $G$. A valid clique ordering is $\{a, b\}, \{c, b, d\}, \{b, d, e\}$. © Ignaz Rutter and Peter Stumpf; licensed under Creative Commons License CC-BY 4.0
Simultaneous Representation of Interval Graphs in the Sunflower Case

90:2

to $\mathcal{R} = (R_1, \ldots, R_k)$ as a simultaneous representation. For two graphs $G, H$ we define the intersection $G \cap H = (V(G) \cap V(H), E(G) \cap E(H))$. A sunflower simultaneous graph is a simultaneous graph $\mathcal{G} = (G_1, \ldots, G_k)$ where $G_1, \ldots, G_k$ have pairwise the same intersection, i.e., there is a graph $S$ with $S = G_i \cap G_j$ for each $i \neq j$. We call $S$ the shared graph of $\mathcal{G}$.

Simultaneous representations have first been studied in the context of graph embeddings where also shared edges have to be represented by the same arc; see [3] for a survey. The notion of simultaneous representations of general intersection graph classes was introduced by Jampani and Lubiw, who gave an $O(n^3)$-algorithm for recognizing simultaneous sunflower permutation graphs (where $n$ is the number of vertices in $G_1 \cup \cdots \cup G_k$) [17], proved NP-completeness for sunflower chordal graphs (as intersection graphs of subtrees of a tree) [17], and gave an $O(n^2 \log n)$-time algorithm for simultaneous interval graphs with $k = 2$ [15]. They left the case $k > 2$ open. The running times of all these algorithms were subsequently reduced to optimal linear time (assuming each input graph is given separately, thus counting the shared graph $k$ times, and assuming that each input graph belongs to the corresponding class) [4, 19].

Since then, the simultaneous representation problem has also been studied for proper and unit interval graphs [20], circle graphs [8] and permutation graphs [17, 19] where $k$ is part of the input. Bok and Jedličková showed that recognizing simultaneous non-sunflower interval graphs is NP-complete [5] if $k$ is part of the input. Similar results hold for simultaneous proper and unit interval representations [20].

A problem closely related to the (sunflower) simultaneous representation problem is partial representation extension, where a representation $R$ of a subgraph $H$ of a single input graph $G$ is given, and the question is, whether $G$ has a representation whose restriction to $H$ coincides with $R$. It has been studied extensively for various graph classes, e.g. for interval graphs [18], circular-arc graphs [10], circle graphs [8, 19], as well as proper and unit interval graphs [18]. Bläsius and Rutter gave a linear-time reduction from the partial interval representation problem to the simultaneous interval representation problem on two graphs [4].

We characterize sunflower simultaneous interval graphs in terms of linear orderings of maximal cliques that satisfy certain consecutivity constrains. This allows us to work with an established data structure called PQ-tree, which represents linear orderings satisfying given consecutivity constraints. The algorithms of Jampani and Lubiw [15] and Bläsius and Rutter [4] for recognizing simultaneous interval graphs with $k = 2$ input graphs use a similar characterization and they also use PQ-trees. Jampani and Lubiw iteratively add non-maximal cliques to orders of maximal cliques and associate nodes of distinct PQ-trees to achieve necessary compatibilities. On the other hand, Bläsius and Rutter synchronize a PQ-tree $T$ that describes orderings of maximal cliques of both input graphs with two PQ-trees $T_1, T_2$ for the two individual input graphs. To this end, they construct PQ-trees for many nodes of $T$ and a 2-SAT formula which describes dependencies between decisions in these trees. While they establish a more general framework for simultaneous PQ-orderings, their approach does not work for more than two input graphs for sunflower simultaneous interval graph recognition.

**Our Result.** We show how to recognize sunflower simultaneous interval graphs in linear time (assuming the input graphs are given separately) even when the number of input graphs is part of the input, thereby answering the open question of Jampani and Lubiw [16]. We note that, similar to Bläsius and Rutter, we use a PQ-tree $T$ that describes orderings of the maximal cliques of the input graphs, synchronize it with PQ-trees for the individual
input graphs, and use a 2-SAT formula to describe dependencies between decisions in these PQ-trees. However, with each operation, they only synchronize pairs of PQ-trees while, in a sense, we synchronize multiple PQ-trees at once. Further, we essentially only construct a single PQ-tree for the synchronization, instead of one for potentially each node of \( T \), which can be linear in the size of the input. For our construction, we exploit a close relation between consecutivity constraints and certain substructures in PQ-trees (Lemma 5) that provides a converse for a natural and widely used property of PQ-trees and may be of independent interest.

**Organization.** In Section 2, we characterize sunflower interval graphs in terms of linear orderings of maximal cliques and we describe PQ-trees. In Section 3, we describe operations on PQ-trees and dependencies between decisions in the original and resulting PQ-trees. In Section 4, we describe our construction, characterize sunflower interval graphs in terms of this construction, and give the linear-time algorithm for the sunflower interval representation problem. In Section 5 we conclude with open questions.

## 2 Preliminaries

For \( n \in \mathbb{N} \) we set \([n] = \{ j \in \mathbb{N} \mid 1 \leq j \leq n \}\). In this paper all graphs are simple.

**Simultaneous Interval Graphs.** An interval representation \( R = \{ I_v \}_{v \in V} \) of a graph \( G = (V, E) \) associates with each vertex \( v \in V \) an interval \( I_v = [x, y] \subseteq \mathbb{R} \) such that for each pair of vertices \( u, v \in V \) we have \( I_u \cap I_v \neq \emptyset \iff \{u, v\} \in E \); see Figure 1. A **simultaneous interval representation** of a simultaneous graph \( G = (G_1, \ldots, G_k) \) assigns each vertex \( v \in \bigcup_{i=1}^k V(G_i) \) an interval \( I_v \) such that the **induced** interval representation \( \{ I_v \}_{v \in V(G)} \) is an interval representation of \( G_i \), for each \( i \in [k] \); see Figure 2. In the following we only consider **sunflower simultaneous graphs** \( G = (G_1, \ldots, G_k) \) which are simultaneous graphs for which there is a graph \( S \) such that \( G_i \cap G_j = S \) for \( i \neq j \). Note that it is necessary that \( S \) is an induced subgraph of each input graph \( G_i \) for \( G \) to be a simultaneous interval graph. We call \( S \) the **shared graph** of \( G \).

It is well known that interval graphs can be characterized via orderings of maximal cliques. A **valid clique ordering** of \( G \) is a linear ordering of the maximal cliques of \( G \) such that for each \( v \in V(G) \) the maximal cliques of \( G \) that contain \( v \) are consecutive.

**Proposition 1** (Fulkerson and Gross [12]). A graph is an interval graph if and only if it admits a valid clique ordering.
Let $\mathcal{G} = (G_1, \ldots, G_k)$ be a sunflower simultaneous graph with shared graph $S$. For $i \in [k]$, let $\mathcal{K}_i$ denote the set of maximal cliques of $G_i$ and let $\mathcal{K} = \bigcup_{i=1}^k \mathcal{K}_i$. Note that we use the disjoint union since we want to treat the maximal cliques of the input graphs separately, even if they coincide with maximal cliques of other input graphs. I.e., each clique in $\mathcal{K}$ is tagged with the input graph $G_i$ it comes from. For a vertex $v \in V(G_i)$, we define $\mathcal{K}_i(v) = \{ C \in \mathcal{K}_i \mid v \in C \}$ as the set of all maximal cliques of $G_i$ that contain $v$ and for $v \in V(S)$, we define $\mathcal{K}(v) = \{ C \in \mathcal{K} \mid v \in C \}$ as the (multi)-set of all maximal cliques of $G_1, \ldots, G_k$ that contain $v$. We further define $\mathcal{K}(S)$ as the set of maximal cliques in the shared graph $S$. A simultaneous clique ordering of $\mathcal{G}$ is a linear ordering $\sigma$ of $\mathcal{K}$ such that the following two properties hold:

- **Property 1.** For each $v \in V(S)$ the set $\mathcal{K}(v)$ is consecutive.
- **Property 2.** The restriction of $\sigma$ to $\mathcal{K}_i$ is a valid clique ordering of $G_i$ for each $i \in [k]$.

The following theorem provides a combinatorial description of sunflower interval graphs which we will use for our algorithm.

- **Theorem 2.** A sunflower simultaneous graph $\mathcal{G}$ is a simultaneous interval graph if and only if it admits a simultaneous clique ordering.

**Proof.** If $\mathcal{G} = (G_1, \ldots, G_k)$ is a simultaneous interval graph, then it has a simultaneous interval representation. For $i \in [k]$, we choose a point $p_C$ for each clique $C \in \mathcal{K}_i$ such that all intervals for $C$ contain $p_C$ and no interval for vertices in $V(G_i) \setminus C$ contains $p_C$. Such a point must exist since intervals have the Helly property and $C$ is maximal in $G_i$. We call these points clique points. Then for each $v \in V(S)$ the clique points in the interval $R(v)$ of $v$ are consecutive and exactly the cliques in $\mathcal{K}(v)$. Note that Property 1 does not necessarily hold for non-shared vertices. However, for each input graph $G_i$ the clique points are placed according to the induced representation of $G_i$ and thus provide a clique ordering of $G_i$.

On the other hand, given a simultaneous clique ordering $\sigma$ for $\mathcal{G}$, we construct an interval representation of $\mathcal{G}$ as follows. We first place distinct (clique) points for the maximal cliques in $\mathcal{K}$ on the real line in the order of $\sigma$ from the left to the right. We then set for each vertex $v \in V$ its interval $R(v)$ to $[l(v), r(v)]$ where $l(v)$ and $r(v)$ are the leftmost point and the rightmost point for cliques containing $v$, respectively. We claim that the clique points and the consecutivity constraints then enforce correct adjacencies. Namely, two vertices $u, v$ of the same input graph $G_i$ have intersecting intervals $R(u)$, $R(v)$ if and only if those intervals share a clique point. It remains to show that $R(u)$, $R(v)$ share a clique point if and only if $u, v$ are adjacent. First observe that if $u, v$ are adjacent, then there is some maximal clique containing both and its clique point is contained in $R(u)$ and $R(v)$ by Properties 1, 2.

For the other direction, let $R(u), R(v)$ share a clique point. We aim to show that there is a clique containing $u$ and $v$, which implies that $u$ and $v$ are adjacent, concluding the proof. Since $R(u)$, $R(v)$ share a clique point, one of the intervals, let’s say $R(u)$, contains an endpoint of the other. That endpoint $p$ is a clique point for a clique $C_p \in \mathcal{K}_i(v)$ by definition of $R(v)$. If $v \notin V(S)$, then $C_p$ is a clique in $G_i$ that also contains $u$ by Property 2. We can argue analogously if $R(v)$ contains an endpoint of $R(u)$ and $u \notin V(S)$. If both $u$ and $v$ lie in $V(S)$, then $C_p$ contains $u$ and $v$ by Property 1. Finally, consider the case where $u \notin V(S)$, $v \in V(S)$ and $R(v)$ contains no end of $R(u)$, meaning that $R(v) \subseteq R(u)$. Since $S$ has a maximal clique $C_v$ containing $v$ and $G_i$ has a maximal clique $C_v'$ containing $C_v$, $R(v)$ must contain a clique point for $C_v'$ and thus $u$ and $v$ are contained in clique $C_v'$.
PQ-Trees. Let $L$ be a set and let $L' \subseteq L$. The ordering $\leq'$ of $L'$ is induced by a linear ordering $\leq$ of $L$, if we have $a \leq' b \Leftrightarrow a \leq b$ for all $a, b \in L'$. A PQ-tree is a data structure that represents linear orderings satisfying a set of consecutivity constraints [6]. Formally, a PQ-tree $T$ on a set $L$ of leaves is a rooted ordered tree where each inner node is either a P-node or a Q-node; see Figure 3. The order of its leaves is the order induced by a preorder traversal of the tree. A PQ-tree $T'$ is equivalent to $T$ if $T'$ can be obtained from $T$ by arbitrarily reordering the children of P-nodes and by reversing the order of the children of any subset of Q-nodes. Note that reversing the order of the children of a Q-node $\lambda$ does not change the order of the children of any child of $\lambda$. In this paper, we consider P-nodes with only two children as Q-nodes. Note that this does not affect which PQ-trees are equivalent. For each inner node $\mu$ we say we flip $\mu$, if we reverse the order of its children.

A PQ-tree represents a linear ordering $\leq$ of $L$ if $\leq$ is the order of the leaves of some equivalent PQ-tree. We write $R(T)$ for all linear orderings of $L$ represented by $T$. The null-tree is defined as a special PQ-tree $T_0$ with $R(T_0) = \emptyset$. For an inner node $\mu$ of a PQ-tree, let $L(\mu)$ denote the leaves of the subtree rooted at $\mu$. For a leaf $\nu$ let $L(\nu) = \{\nu\}$. We denote the lowest common ancestor of a set $N \subseteq V(T)$ by $\text{lca}_T(N)$. We say that a node $\nu$ of $T$ is left of a node $\lambda$ in $t$ if $\nu$ comes before $\lambda$ in a preorder traversal. However, the order of a node and one of its ancestors will never be relevant in this paper.

3 Operations on PQ-Trees

By Theorem 2, we can recognize sunflower simultaneous interval graphs by testing the existence of a simultaneous clique ordering. We use PQ-trees to describe clique orderings that satisfy the properties of a simultaneous clique ordering. Namely, we use a PQ-tree $T$ with leaf set $K$ to describe all linear orderings satisfying Property 1 and PQ-trees $T'_1, \ldots, T'_k$ where each $T'_i$ represents all valid clique orderings of $G_i$. A simultaneous clique ordering is then a linear ordering $\sigma \in R(T)$ that induces orderings in $R(T'_1), \ldots, R(T'_k)$.

To find such a linear ordering $\sigma$, we “synchronize” these PQ-trees. One step will be to construct a PQ-tree $T_S$ on $K(S)$ that describes the maximal clique orderings of $S$ that are in some sense compatible with the linear orderings in $R(T'_1), \ldots, R(T'_k)$. Another aspect is to describe the dependencies between decisions at Q-nodes in all constructed PQ-trees.

Consistency and Backward-Consistency. We say that an ordered pair of leaves $(\lambda_1, \lambda_2)$ is forward directed if $\lambda_1$ comes before $\lambda_2$ in the leaf ordering of the PQ-tree. Otherwise, we call it backward directed. Hence, a pair of leaves is either forward directed or backward directed. Note that the corresponding children $\nu_1, \nu_2$ of $\text{lca}(\lambda_1, \lambda_2)$ with $\lambda_1 \in L(\nu_1)$ and $\lambda_2 \in L(\nu_2)$
are ordered the same way as $\lambda_1, \lambda_2$. Hence, if $\mu$ is a Q-node, flipping $\mu$ changes the order of $\lambda_1, \lambda_2$. Let $T_1, T_2$ be two PQ-trees on sets $L_1, L_2$ with $L_1 \subseteq L_2$ and let $\mu_1, \mu_2$ be two Q-nodes in $T_1, T_2$ such that there are two leaves $\lambda_1, \lambda_2$ whose order is affected by flipping $\mu_1$ or $\mu_2$, respectively. More formally, let $\mu_1$ be a Q-node in $T_1$ with children $\nu_1^1, \nu_1^2$ and let $\mu_2$ be a Q-node in $T_2$ with children $\nu_2^1, \nu_2^2$, such that there exist two leaves $\hat{\lambda}_1 \in L(\nu_1^1) \cap L(\nu_1^2)$ and $\hat{\lambda}_2 \in L(\nu_2^1) \cap L(\nu_2^2)$.

We say $\mu_1$ and $\mu_2$ are consistent, if for each pair $(\lambda_1, \lambda_2)$ of leaves with $\mu_1 = \text{lca}_{T_1}(\lambda_1, \lambda_2)$, $\mu_2 = \text{lca}_{T_2}(\lambda_1, \lambda_2)$ we have that $(\lambda_1, \lambda_2)$ is forward directed in $T_1$ if and only if it is forward directed in $T_2$. We say $\mu_1$ and $\mu_2$ are reverse consistent, if for each pair $(\lambda_1, \lambda_2)$ of leaves with $\mu_1 = \text{lca}_{T_1}(\lambda_1, \lambda_2)$, $\mu_2 = \text{lca}_{T_2}(\lambda_1, \lambda_2)$ we have that $(\lambda_1, \lambda_2)$ is forward directed in $T_1$ if and only if $(\lambda_1, \lambda_2)$ is backward directed in $T_2$. Observe that $\mu_1, \mu_2$ cannot be both consistent and reverse consistent at the same time. Consider the case where $\mu_1, \mu_2$ are neither consistent nor reverse consistent. Then $\mu_1, \mu_2$ order at least one pair of leaves differently (one forward and one backward directed) and at least one pair of leaves the same way (forward or backward directed). This remains true even after flipping one or both of $\mu_1, \mu_2$. Hence, in that case no linear order in $R(T_1)$ can be extended to a linear order in $R(T_2)$, since at least one pair of leaves is ordered differently.

We use four operations on PQ-trees for the construction of the PQ-tree $T_S$ that orders the maximal cliques of the shared graph: reduction, intersection, projection and pruning. While the first three operations are frequently used for PQ-trees, pruning is less common, but was for example used in the context of level planarity [7] where the algorithm is attributed to Di Battista and Nardelli [2].

To achieve a linear running time for our algorithm, we track what happens to Q-nodes when applying these operations in a PQ-tree, similar as in [4].

**Reduction.** Let $T$ be a PQ-tree with leaf set $L$. The reduction of $R(T)$ with a set $L' \subseteq L$ is the set $R'(T, L')$ of linear orderings in $R(T)$ where $L'$ is consecutive. A PQ-tree $T'$ with $R(T') = R'(T, L')$ can be computed from $T$ in $O(|L'|)$ time [6]. This operation is the main operation for PQ-trees, since it allows to compute a PQ-tree $\hat{T}$ on $L$ where sets $S_1, \ldots, S_k \subseteq L$ are consecutive efficiently.

**Proposition 3 (Booth and Lueker [6]).** Let $L$ be a finite set and let $S_1, \ldots, S_k \subseteq L$ be non-empty sets. A PQ-tree $\hat{T}$ on $L$ that represents the linear orderings of $L$ where $S_1, \ldots, S_k$ are consecutive can be computed in $O(|L| + \sum_{i=1}^{k} |S_i|)$ time.

While the reduction for PQ-trees was originally described by applying a variety of templates, Hsu gave an alternative description [14]; see also [11]. Roughly speaking, we find a certain path $P$ in $T$ that separates $L'$ and $L \setminus L'$, split P-nodes on $P$ suitably and merge the resulting nodes on $P$ to a single P-node. Finally we remove some degeneracies (especially, if $P$ consists of a single P-node $\lambda$, the resulting Q-node is smoothed, effectively just splitting $\lambda$ into two P-nodes); see Figure 3. For more details, we refer to the papers of Booth and Lueker [6] or Hsu [14]. We only use the running time result for the construction of PQ-trees satisfying given consecutivity constraints mentioned above. Especially, we do not need to keep track of the consistencies between Q-nodes for the reduction.

**Intersection.** Let $T_1, T_2$ be PQ-trees with leaf set $L$. The intersection $T_1 \cap T_2$ is a PQ-tree on $L$ representing $R(T_1) \cap R(T_2)$. It can be computed from $T_1, T_2$ in $O(|L|)$ time together with the consistencies between the Q-nodes in $T_1 \cap T_2$ and the Q-nodes in $T_1$ and $T_2$ [19].

For any two cliques $A, B \in L$ where $\text{lca}_{T_1}(A, B)$ or $\text{lca}_{T_2}(A, B)$ is a Q-node, one can see that $\text{lca}_{T_1 \cap T_2}(A, B)$ is a Q-node as follows. Let $\text{lca}_{T_1}(A, B)$ be a Q-node. We can obtain $T_1 \cap T_2$ from $T_1$, by applying a reduction on $T_1$ for each consecutivity constraint of $T_2$. Since
the only possible change to Q-nodes in reductions is being merged with other nodes to a Q-node of higher degree, the lowest common ancestor of $A, B$ remains a Q-node (Note that the reduction of a PQ-tree for a given set is unique up to equivalence).

**Projection.** Let $T$ be a PQ-tree with leaf set $L$ and let $L' \subseteq L$. The projection $R^*(T, L')$ from $R(T)$ to $L'$ is the set of linear orderings of $L'$ induced by the linear orderings in $R(T)$ on $L'$. The projection $T'$ of $T$ on $L'$ is a PQ-tree on $L'$ with $R(T') = R^*(T, L')$. It can be computed in $O(|L|)$ time from $T$ by only keeping nodes $\mu$ with $L(\mu) \cap L' \neq \emptyset$ and smoothing all nodes with a single child (that is, removing the node and adding an arc connecting its parent with its child). Here all Q-nodes in $T'$ are consistent to their respective copy in $T$.

**Prune.** Let $T$ be a PQ-tree with leaf set $L$, let $\ell' \notin L$ and let $L' \subseteq L$ be consecutive in each $\sigma \in R(T)$. For any $\sigma \in R(T)$, the prune of $L'$ to $\ell'$ in $\sigma$ is the result of replacing $L'$ in $\sigma$ by $\ell'$. The prune of $L'$ to $\ell'$ in $R(T)$ is the set containing for each $\sigma \in R(T)$ the prune of $L'$ to $\ell'$ in $\sigma$. For pruning PQ-trees, we first observe that consecutive sets of leaves correspond to simple substructures of PQ-trees.

**Lemma 4.** Let $T$ be a PQ-tree with leaf set $L$ and let $L' \subseteq L$ be consecutive in each $\sigma \in R(T)$. Then, there is either a P-node $\lambda$ with $L(\lambda) = L'$ or a Q-node $\mu$ with a consecutive subset of children $\nu_1, \ldots, \nu_l$ such that $\bigcup_{i=1}^l L(\nu_i) = L'$.

**Proof.** We consider $\nu = \text{lca}(L')$, i.e., $L' \subseteq L(\nu)$ and there are two distinct children $\mu_1, \mu_2$ of $\nu$ with $L' \cap L(\mu_1) \neq \emptyset$ and $L' \cap L(\mu_2) \neq \emptyset$.

First assume $\nu$ is a P-node. Then for any other child $\xi$ of $\nu$ we have $L(\xi) \subseteq L'$ since otherwise we could violate the consecutivity of $L'$ by placing $\xi$ between $\mu_1$ and $\mu_2$. We further have $L(\mu_1) \subseteq L'$ and $L(\mu_2) \subseteq L'$, since otherwise the consecutivity of $L'$ is violated after flipping $\mu_1$ and $\mu_2$. Hence, we have $L(\nu) = L'$.

Next let $\nu$ be a Q-node, and let $\mu_1, \mu_2$ be the leftmost and the rightmost child of $\nu$ with descendants in $L'$. Since $L'$ is consecutive, all children between $\mu_1$ and $\mu_2$ only have descendants in $L'$. If $\mu_1$ or $\mu_2$ had a descendant not in $L'$, then the consecutivity of $L'$ could be violated by flipping it. Hence, $\nu$ has the stated property.

With this insight, the prune of $L'$ to $\ell'$ in $T$ is obtained as follows. By Lemma 4, either $\text{lca}(L')$ is a P-node with $L(\text{lca}(L')) = L'$ or $\text{lca}(L')$ is a Q-node with consecutive children $\nu_1, \ldots, \nu_l$ such that $L' = \bigcup_{i=1}^l L(\nu_i)$. If $\text{lca}(L')$ is a P-node, the prune is obtained by replacing $\text{lca}(L')$ and its subtree by leaf $\ell'$. Otherwise, the prune is obtained by replacing $\nu_1, \ldots, \nu_l$ by $\ell'$ as a child of the Q-node $\text{lca}(L')$. Clearly, given $L'$, the prune can be computed in $O(|L'|)$ time from $T$ with a bottom-up approach. We introduce additional consistencies. Namely, we consider each Q-node $\mu$ of $T$ consistent to its copy $\mu'$ in the prune of $L'$ to $\ell'$ in $T$, if that copy exists. These consistencies are trivial and not explicitly computed.

**4 Recognition Algorithm**

We use Theorem 2 to recognize sunflower simultaneous interval graphs by deciding whether a given sunflower simultaneous graph $G$ has a simultaneous clique ordering. The rough idea is the following. For Property 1, we construct a PQ-tree $T$ on $K$ where $K(\nu)$ is consecutive for each $\nu \in V(S)$. For Property 2, we construct for each $i \in [k]$ a PQ-tree $T'_i$ on $K_i$ where $K_i(\nu)$ is consecutive for each $\nu \in V(G_i)$. I.e., each $T'_i$ represents the valid clique orderings of $G_i$; see Figure 4. By construction, a simultaneous clique ordering of $G$ is then a linear ordering $\sigma \in R(T)$ that induces a linear ordering in each of $R(T'_1), \ldots, R(T'_k)$.
Simultaneous Representation of Interval Graphs in the Sunflower Case

\[ \begin{array}{cccc}
    r & s & t & u \\
    a & b & c & y
\end{array} \]

\[ T \]

\[ T' \]

\[ T_1 \]

\[ T_2 \]

\[ T_1' \]

\[ T_2' \]

\[ T_1|_S \]

\[ T_2|_S \]

\[ T_S \]

Figure 4 Top: Simultaneous representation of a sunflower simultaneous graph \( G = (G_1, G_2) \) with \( V(G_1) = \{a, b, c, r, s, t, u\} \) and \( V(G_2) = \{x, y, r, s, t, u\} \). Below are the PQ-trees for \( G \) with circles for P-nodes and squares for Q-nodes, as defined in Section 4. The leaves are cliques described as strings of the contained vertices.
This means that we obtain $\sigma$ as the order of leaves of $T$, if for $i \in [k]$ any two cliques $A, B \in K_i$ are ordered the same way in the order of leaves of $T$ and in the order of leaves of $T'_i$. We construct PQ-trees $T_1, \ldots, T_k$ by restricting $T'_1, \ldots, T'_k$ such that they are “compatible” with $T$ while only loosing linear orderings that do not match any simultaneous clique ordering. We will show that if $lca_T(A, B)$ is a Q-node, then so is $lca_T(A, B)$. This allows to ensure the same ordering of $A, B$ in $T$ and $T'_i$ by making each pair of backward-consistent Q-nodes consistent. We achieve this with a 2-SAT formula $\Phi$. If $\mu = lca_T(A, B)$ is a P-node, then we cannot just arrange the children of $\mu$ according to $T_i$, since this can also affect the order of maximal cliques for other input graphs. However, we can resolve this problem as follows. We will see that, if $A$ or $B$ is a child of $\mu$ directly, then its position can be chosen according to $T_i$ without affecting the order of the cliques for other input graphs. Otherwise, $\mu$ has two inner nodes $\nu_1, \nu_2$ with $A \in L(\nu_1), B \in L(\nu_2)$ as children. The next lemma then shows that the order of $\nu_1$ and $\nu_2$ is in a sense decided by the order of two shared intervals (“below” $\nu_1$ and $\nu_2$, respectively). This allows us to synchronize $T$ and $T_1, \ldots, T_k$ by considering corresponding valid clique orderings for $K(S)$. Namely, we use the operations on PQ-trees from Section 3 to obtain a PQ-tree $T'$ on $K(S)$ that is in a sense compatible with $T_1, \ldots, T_k$ and $T$. In $T$, we order the children of a P-node $\mu$ that are inner nodes according to the order of corresponding shared intervals whose order is given by the order of the leaves of $T_S$, and we apply corresponding orderings in each $T_i$. This ensures compatibility of the orders of $T_1, \ldots, T_k$, since all children of $\mu$ that are leaves are private to some $T_i$ and can be arranged accordingly in $T$.

Note that each consecutivity constraint for $T$ is a set $K(v)$ with $v \in V(S)$. In that sense, by the following lemma each child of a P-node in $T$ has a private shared vertex $v \in V(S)$ if it is an inner node.

Lemma 5. Let $L$ be a finite set and let $\{S_1, \ldots, S_k\} \subseteq 2^L$ with $|S_i| \geq 2$ for $i \in [k]$. Let $T$ be the PQ-tree on $L$ obtained by making $S_1, \ldots, S_k$ consecutive. Let $\mu$ be a P-node and let $\nu$ be a child of $\mu$ that is not a leaf. Then if $\nu$ is a P-node, there is an $S_i$ with $L(\nu) = S_i$. If $\nu$ is a Q-node, there is an $S_i$ with $\bigcup_{j=1}^k L(\nu_j) = S_i$ for a consecutive subset of children $\nu_1, \ldots, \nu_l$ of $\nu$.

Proof. Let $\nu$ be a P-node and suppose there is no $S_i$ with $L(\nu) = S_i$. First observe that by Lemma 4, for any $S_i$ with $S_i \cap L(\mu) \neq \emptyset$, we have either $L(\mu) \subseteq S_i$ or $S_i \subseteq L(\lambda)$ for some child $\lambda$ of $\mu$. This means that, after contracting the arc $\mu \nu$, no $S_i$ can be violated. However, the contraction allows to order other children of $\mu$ between the children of $\nu$. Thereby $L(\nu)$ is no longer consecutive in all represented linear orderings. This contradicts $T$ originally representing all linear orderings where $S_1, \ldots, S_k$ are consecutive, since in each $\sigma \in R(T)$ for the original $T$, leaf set $L(\nu)$ is consecutive.

Next, let $\nu$ be a Q-node and suppose there is no $S_i$ with $\bigcup_{j=1}^k L(\nu_j) = S_i$ for any consecutive subset of children $\nu_1, \ldots, \nu_l$ of $\nu$. If $\nu$ has precisely two children, we treat it as a P-node and argue as above that there is an $S_i$ with $L(\nu) = S_i$. Hence, assume that $\nu$ has at least three children. By Lemma 4, for any $S_i$ with $S_i \cap L(\mu) \neq \emptyset$, we then have either $L(\mu) \subseteq S_i$ or $S_i \subseteq L(\lambda)$ for some child $\lambda$ of $\mu$. This means that after switching the label of $\mu$ from Q-node to P-node, still all represented linear orderings have all $S_i$ consecutive. This contradicts the choice of $T$, since by making $\mu$ a P-node, $T$ represents additional linear orderings.

Note that Lemma 5 is in a sense the converse of Lemma 4 for the children of P-nodes.
4.1 Polynomial-Time Algorithm

We now describe the construction of the 2-SAT formula $\Phi$ and all relevant PQ-trees. For $\Phi$, each Q-node $\lambda$ (of any constructed PQ-tree) is assigned a Boolean variable $x_\lambda$ that tells whether it should be flipped, and we add $(x_\lambda \leftrightarrow x_\mu)$ for consistent nodes $\mu$ and we add $(x_\lambda \not\leftrightarrow x_\nu)$ for backward-consistent nodes $\nu$ to $\Phi$. We describe for which PQ-trees we need to consider the consistencies after introducing all PQ-trees.

Let $G = (G_1, \ldots, G_k)$ be a sunflower simultaneous graph and let $T$ be the PQ-tree on $K$ that enforces consecutivity of each set $\mathcal{K}(v)$ with $v \in V(G)$; see Figure 4. Note that if $T$ is the null-tree, then there exists no simultaneous clique ordering since Property 1 cannot be satisfied, and by Theorem 2 there is no simultaneous interval representation of $G$. Hence, we assume in the following that $T$ is not the null-tree.

For $i \in [k]$, let $T_i$ be the projection of $T$ on $K_i$, and let $T_i'$ be the PQ-tree on $K_i$ that enforces consecutivity of each set $\mathcal{K}_i(v)$ with $v \in V(G_i)$. Note that $T_i'$ describes all valid clique orderings of $G_i$. We are interested in the PQ-tree $T_i = T_i' \cap T_1'$, which restricts the valid clique orderings of $G_i$ to those that are compatible with Property 1; see Figure 4. With Property 2 this means that, if any $T_i$ is the null-tree, then there is no simultaneous clique order. Hence, we assume in the following that no $T_i$ is the null-tree.

We would like to synchronize $T_1, \ldots, T_k$ with $T$. However, they have distinct leaf sets. Thus, we cannot just intersect them. Instead, we aim to describe the clique orderings for $S$ that can be induced by $T_1, \ldots, T_k$ with PQ-trees. This allows us to find a clique ordering for $S$ that is compatible with all $T_1, \ldots, T_k$. With the Q-nodes flipped according to a solution of $\Phi$, this will be enough to synchronize $T_1, \ldots, T_k$ with $T$.

We next aim to prune $T_1, \ldots, T_k$ to maximal cliques of $S$. For any clique $A \in \mathcal{K}(S)$, we define $\mathcal{K}_i(A)$ as the set of maximal cliques of $G_i$ that contain $A$ as a subclique. It is $\mathcal{K}_i(A) = \{C \in \mathcal{K}_i \mid A \subseteq C\} = \bigcap_{v \in A} \{C \in \mathcal{K}_i \mid v \in C\} = \bigcap_{v \in A} (\mathcal{K}(v) \cap \mathcal{K}_i)$. The critical observation is that, since the intersection of consecutive sets is itself consecutive in a linear order, $\mathcal{K}_i(A)$ is consecutive in $T_i'$ and thus in $T_i$. Note that for distinct $A, B \in \mathcal{K}(S)$, the sets $\mathcal{K}_i(A)$ and $\mathcal{K}_i(B)$ are disjoint, since the set of shared vertices in a maximal clique $C \in \mathcal{K}_i(A) \cap \mathcal{K}_i(B)$ would otherwise be $A$ as well as $B$.

We can now construct for each $T_i$ a PQ-tree describing the corresponding orderings of $\mathcal{K}(S)$ as follows. For $i \in [k]$, let $T_i^*$ be the PQ-tree obtained by starting with $T_i$ and pruning for each $A \in \mathcal{K}(S)$ the set $\mathcal{K}_i(A)$ to leaf $A$. Then, let $T_i|_S$ be the projection of $T_i^*$ on $\mathcal{K}(S)$.

Finally, let $T_S = \bigcap_{i=1}^k T_i|_S$ be the intersection of all $T_i|_S$; see Figure 4. By construction, $R(T_S)$ contains all clique orderings of $S$ that can be induced by a simultaneous clique order. Hence, if it is the null-tree, there is no simultaneous clique order.

For each $1 \leq i \leq k$, we add clauses to $\Phi$ for the consistencies between $T$ and $T_i$, between $T_i$ and $T_1^*$, between $T_i$ and $T_i^*$, between $T_1^*$ and $T_i|_S$, and between $T_i|_S$ and $T_S$. If $\Phi$ is not satisfiable, then there is no simultaneous clique ordering. On the other hand, with this, the necessary conditions are also sufficient.

**Theorem 6.** $(G_1, \ldots, G_k)$ is a sunflower interval graph if and only if $\Phi$ is satisfiable and neither $T$ nor $T_S$ is the null-tree.

**Proof.** If $(G_1, \ldots, G_k)$ is a sunflower interval graph the requirements are necessary as discussed above. Hence, assume that neither $T$ nor $T_S$ is the null-tree and that $\Phi$ has a satisfying assignment $\Gamma$. By Theorem 2, it suffices to find a simultaneous clique ordering. We aim to operate on $T$ and each $T_i$ such that the order $\sigma$ of $T$ induces the order of each $T_i$, thus ensuring that $\sigma$ is a simultaneous clique ordering. We first flip all Q-nodes according to $\Gamma$ (that is, flip each Q-node $\lambda$ where $x_\lambda$ is true). This ensures that any two cliques $A, B$
in $K$ or $C(S)$ are ordered the same way in each of $T, T_1, \ldots, T_k$ or $T_i|s, \ldots, T_k|s, T_S$ where $\text{lca}(A, B)$ is a Q-node and the sets $K_i(A), K_i(B)$ are ordered in $T_i$ the same way as $A, B$ are ordered in $T_i|s, i \in [k]$.

We next order the children of the P-nodes of $T$. Let $\mu$ be a P-node of $T$. Then, by Lemma~5 for each child $\nu$ of $\mu$ that is an inner node, there is a vertex $v \in S$ such that $K(\nu) \subseteq L(\nu)$: We choose an arbitrary clique $C_\nu \in K(S)$ that contains $v$. We then order the children $\nu$ of $\mu$ that are inner nodes according to the order of the corresponding cliques $C_\nu$ given by $T_S$. For $i \in [k]$, we order the sets $K_i(C_\nu)$ in $T_i$ accordingly. First note that these sets are not empty since each $G_i$ contains a clique containing $C_\nu$. Next note that this orders any two leaves $\lambda_1, \lambda_2 \in K_i$ with $\mu = \text{lca}_T(\lambda_1, \lambda_2)$ that are not children of $\mu$ the same way in $T_i$ as in $T$ since for the corresponding children $\nu_1, \nu_2$ of $\mu$ we have that $L(\nu_1)$ and $L(\nu_2)$ are consecutive in $T$ and thus also in $T_i$. E.g., even if $\nu_1$ is a Q-node and $K_i(C_{\nu_1})$ does not contain $\lambda_1$, this rearrangement still ensures the correct ordering of $\lambda_1$ and $\lambda_2$ in $T_i$. Finally note that this does not flip any Q-nodes of $T_i$ since projection and intersection preserve Q-nodes that order two leaves of the projection set~[4]. I.e., for each pair of cliques $A, B \in K(S)$ such that the order of $K_i(A), K_i(B)$ is decided by a Q-node $\mu$ in $T_i$, there is a Q-node in $T_S$ deciding the order of $A, B$ the same way as $\mu$ orders $K_i(A), K_i(B)$ (after flipping Q-nodes according to $\Gamma$).

With this, for any P-node $\mu$ of $T$ and any two children $\nu_1, \nu_2$ of $\mu$ that are inner nodes, each $T_i$ orders any pair of $A \in L(\nu_1) \cap K_i$ and $B \in L(\nu_2) \cap K_i$ the same way as $T$. This allows us to order the children of $\mu$ simultaneously according to $T_1, \ldots, T_k$ where each inner node $\nu$ is ordered as any leaf $C \in L(\nu) \cap K_i$. Note that each child of $\mu$ that is a leaf is contained in a single $T_i$, i.e., it can be placed solely considering the order in $T_i$ (which ensures the correct order with regards to the children of $\mu$ that are inner nodes). Since $L(\nu) \cap K_i$ is consecutive in $T_i$, the choice of $C(\nu)$ does not matter. I.e., we find an ordering of all children of $\mu$, which is compatible with the orderings given by $T_1, \ldots, T_k$. It remains to show that $T$ now actually provides a simultaneous clique ordering. Property 1 is satisfied by the definition of $T$. For Property 2 we verify that the order of $K_i$ induced by $T$ is the same as the one given by $T_i$. Consider any two cliques $A, B \in K_i$. If $\text{lca}_T(A, B)$ is a Q-node, then $A, B$ are ordered the same way in $T$ and $T_i$, since we flipped the Q-nodes according to $\Phi$. If $\text{lca}_T(A, B)$ is a P-node, then they are ordered the same way in $T$ and $T_i$ by the operations we just applied on the P-nodes of $T$.

All three requirements in Theorem~6 are necessary; see Figure~5. Theorem~6 allows to recognize sunflower interval graphs in polynomial time by constructing $T, T_S$ and $\Phi$. If $G$ is a simultaneous interval graph, we obtain a simultaneous clique ordering of $G$ by following the construction in the proof of Theorem~6. With the construction in Theorem~2, we then obtain a simultaneous interval representation.

\begin{corollary}
Sunflower interval graphs can be recognized in polynomial time. For yes-instances a simultaneous interval representation can also be constructed in polynomial time.
\end{corollary}

\subsection{Linear-Time Algorithm}

To achieve a linear running time, we use that the construction steps can be done efficiently, while also computing the consistencies between Q-nodes, as discussed in Section~3. However, we cannot afford to compute the projection from $T$ on $K_i$ for each $i \in [k]$ separately, since this could result in an additional factor of $k$ for the running time. Instead, we use the next lemma to compute the projections simultaneously. A similar argumentation has been used by Münch et al. [19].
Simultaneous Representation of Interval Graphs in the Sunflower Case

Figure 5 Variants of a sunflower graph $G = (G_1, G_2)$ where $G_1$ and $G_2$ are interval graphs, but $G$ is not a simultaneous interval graph. a) $T$ is a null-tree since the sets \{abc, bx\}, \{abc, ay\} and \{abc, cz\} have to be consecutive, while abc can only have two neighbors in the linear ordering. b) $T_S$ is a null-tree since $G_1$ forces $b$ to be in the middle of $a$ and $c$, while $G_2$ forces $a$ to be in the middle of $b$ and $c$. c) $\Phi$ cannot be satisfied since $\xi$ is consistent to $\mu$ and $\nu$ while $\lambda$ is consistent to $\mu$ and backward-consistent to $\nu$ (note that these consistencies are implied in $\Phi$ via the other constructed PQ-trees).
Lemma 8. Let $T$ be an ordered tree with leaf set $L$ and let $S = \{S_1, \ldots, S_i\} \subseteq 2^L$. Then, for each $S_i \in S$ the projection $T^*_i$ of $T$ on $S_i$ can be computed such that each node of $T^*_i$ holds a reference to its original copy in $T$, with a total running time in $O(|L| + \sum_{i=1}^t |S_i|)$.

Proof. Observe that a preorder traversal of a $T^*_i$ is a subsequence of a preorder traversal of $T$. We create a list $U$ that contains a tuple $(S_i, \lambda, P)$ where $\lambda, p$ is the position of $\lambda$ in the preorder of $T$, for each $S_i \in S$ and $\lambda \in S_i$. We then sort $U$ lexicographically in linear time using radix sort [9]. Then, for each set $S_i \in S$, the tuples with $S_i$ are consecutive and provide the order of the leaves in $S_i$ in the preorder traversal. For any $S_i$, we find all nodes of $T^*_i$ in $T$ with the following observation. Let $\mu$ be a node of $T^*_i$ and let $\nu_1, \nu_2$ be two children of $\mu$ with $\nu_1 < \nu_2$ in the preorder. Then $\mu$ is the lowest common ancestor of the rightmost leaf in $L(\nu_1)$ and the leftmost leaf in $L(\nu_2)$. Hence, each inner node of $T^*_i$ is the lowest common ancestor of two consecutive leaves. We use the lowest common ancestor data structure for static trees by Harel and Tarjan [13], to compute for every pair of nodes $\lambda_1, \lambda_2$ in $S_i$ that are consecutive in the preorder the least common ancestor (and descendant of the other one). By removing $S_i$ and then all duplicates from $U$, we get a list of all nodes of $S_i$ of height 1. Note that all duplicates of a node $\lambda$ are consecutive, when they are removed. By repeating the same for each height, we get the parents of all nodes.

With that, the construction of the PQ-trees is straightforward.

Corollary 9. Sunflower interval graphs can be recognized in $O(\sum_{i=1}^k (|V(G_i)| + |E(G_i)|))$ time, where $(G_1, \ldots, G_k)$ is the input sunflower graph. For yes-instances a simultaneous interval representation can be constructed in the same asymptotic running time.

Proof. We follow the construction of $T, T_1, \ldots, T_k, T_1|S, \ldots, T_k|S, T_S$ for Theorem 6. For each constructed PQ-tree, we maintain the consistencies to the PQ-tree(s) it is constructed from (except for consistencies to unchanged copies of a Q-node, where we use the same variable). The trees $T$ and $T_1', \ldots, T_k'$, can be constructed in $O(\sum_{i=1}^k (|V(G_i)| + |E(G_i)|))$ time by Proposition 3. Then, $T_1', \ldots, T_k'$ can be constructed in $O(\sum_{i=1}^k (|V(G_i)| + |E(G_i)|))$ time by Lemma 8. The PQ-trees $T_1, \ldots, T_k$ can be constructed in the same total time [19].

$T_1|S, \ldots, T_k|S$ can then be constructed in $O(\sum_{i=1}^k (|V(G_i)| + |E(G_i)|))$ by computing the projection and prunes directly. However, we store the smoothed nodes and pruned subtrees (or sets of subtrees), such that we can compute easily a linear order in $R(T_i)$ whose prune is a given linear order in $R(T_i|S)$ (after projection to $K(S)$). Finally, $T_S$ can be computed in $O(k \cdot (|V(S)| + |E(S)|))$ time. The 2-SAT formula $\Phi$ can easily be computed from the maintained consistencies. A solution for $\Phi$ can be computed in linear time [1]. With Theorem 6 this suffices to decide whether there is a simultaneous interval representation.

To compute such a representation in linear time, we construct the simultaneous clique ordering a bit differently than in Theorem 6. We first operate on $T_1|S, \ldots, T_k|S$ such that their leaves are ordered as in $T_S$. This can be done in $O(k \cdot (|V(S)| + |E(S)|))$ time. Then we reverse the smoothing and pruning from $T_1, \ldots, T_k$ (using the stored nodes and subtrees) to obtain corresponding linear orderings in $R(T_1), \ldots, R(T_k)$. This can be done in time linear in the size of $T_1, \ldots, T_k$, i.e., in $O(\sum_{i=1}^k (|V(G_i)| + |E(G_i)|))$ time.

In the proof of Theorem 6 we established that the obtained linear orderings can now be merged to a simultaneous clique ordering. We only need to ensure the consecutivities for the shared vertices. Then, we compute for $i \in [k]$ the first and last position $s_i^\prime$, $t_i^\prime$ of each
shared vertex \( v \in V(S) \) in the clique ordering of \( T_i \) by iterating over the clique ordering once. For \( i \in [k] \), we sort a list of all these positions in \( O(|V(G)| + |E(G)|) \) time using counting sort [9]. After removing duplicates of positions appearing multiple times, we assign each shared vertex \( v \) the positions \( s^v_i, t^v_i \) of \( s^v, t^v \) in that sorted list. We can then consider for each vertex \( v \in V(S) \) two \( k \)-tuples \( s^v = (s^v_1, \ldots, s^v_k) \) and \( t^v = (t^v_1, \ldots, t^v_k) \). We sort a list \( L_S \) of all these \( k \)-tuples lexicographically in \( O(k \cdot |V(S)|) \) time using radix sort [9]. This provides us with an order of the start and endpoints of the intervals of the shared vertices in a simultaneous interval representation.

We get a simultaneous clique ordering \( \sigma \) by simultaneously iterating over the sorted list \( L_S \) and the clique orderings of \( T_1, \ldots, T_k \) as follows. At each entry \( s^v \) (or \( t^v \)) of \( L_S \), append to \( \sigma \) for each \( T_i \) all cliques between the last appended clique and position \( s^v_i \) (or \( t^v_i \)). After the last entry of \( L_S \), append the remaining cliques to \( \sigma \). Since we followed the construction of the proof of Theorem 6 there is a simultaneous clique ordering inducing the clique orderings of \( T_1, \ldots, T_k \). Thus, we have for any two entries \( r = (r_1, \ldots, r_k), r' = (r'_1, \ldots, r'_k) \) that \( r \leq r' \) in \( L_S \) only if \( r_i \leq r'_i \), for all \( i \in [k] \). This ensures that each clique \( C \) is appended when \( C \cap V(S) \) are precisely the shared vertices \( v \) for which \( s^v \) is processed, but \( t^v \) is not. Hence, Property 1 is satisfied and \( \sigma \) actually is a simultaneous clique ordering. With that a simultaneous interval representation can be computed straightforwardly, by placing a point for each clique on the real line in that order and then assigning to each vertex \( v \) the interval \([s_v, t_v]\) where \( s_v, t_v \) are the points for the first and last clique containing \( v \), as done for Theorem 2.

5 Open Questions

While we solve the sunflower representation problem for interval graphs in linear time if each input graph is given separately, a more compact input description is possible, if the number of non-shared vertices is very small. In that case, the input can be given as the union graph \( G = \bigcup_{i=1}^{k} G_i \) as a single graph with labels describing which input graph a non-shared vertex belongs to. Our approach would then have a running time in \( O(k \cdot (|V(G)| + |E(G)|)) \).

▶ Question 1. Can the sunflower representation problem for interval graphs be solved in \( o(k \cdot (|V(G)| + |E(G)|)) \) time if the input is given as the union graph \( G \)?

Note that it is not clear how to even verify that each input graph is an interval graph with less time.

While the general simultaneous representation problem is NP-complete for interval graphs if the number of input graphs is part of the input, and it is solvable in linear time for \( k = 2 \), we do not know the complexity for fixed \( k > 2 \).

▶ Question 2. Can the simultaneous representation problem for interval graphs be solved in polynomial time for a fixed \( k > 2 \)? In particular, considering \( k \) as a parameter, is the problem in \( XP \)? Is it \( FPT \)?

References


