

Simultaneous Interval graphs

Krishnam Raju Jampani *

Anna Lubiw †

Abstract

In a recent paper, we introduced the simultaneous representation problem (defined for any graph class \mathcal{C}) and studied the problem for chordal, comparability and permutation graphs. For interval graphs, the problem is defined as follows. Two interval graphs G_1 and G_2 , sharing some vertices I (and the corresponding induced edges), are said to be “simultaneous interval graphs” if there exist interval representations R_1 and R_2 of G_1 and G_2 , such that any vertex of I is mapped to the same interval in both R_1 and R_2 . Equivalently, G_1 and G_2 are simultaneous interval graphs if there exist edges E' between $G_1 - I$ and $G_2 - I$ such that $G_1 \cup G_2 \cup E'$ is an interval graph.

Simultaneous representation problems are related to simultaneous planar embeddings, and have applications in any situation where it is desirable to consistently represent two related graphs, for example: interval graphs capturing overlaps of DNA fragments of two similar organisms; or graphs connected in time, where one is an updated version of the other.

In this paper we give an $O(n^2 \log n)$ time algorithm for recognizing simultaneous interval graphs, where $n = |G_1 \cup G_2|$. This result complements the polynomial time algorithms for recognizing probe interval graphs and provides an efficient algorithm for the interval graph sandwich problem for the special case where the set of optional edges induce a complete bipartite graph.

Keywords: Simultaneous Graphs, Interval Graphs, Graph Sandwich Problem, Probe Graphs, PQ-trees

1 Introduction

Let \mathcal{C} be any intersection graph class (such as interval graphs or chordal graphs) and let G_1 and G_2 be two graphs in \mathcal{C} , sharing some vertices I and the edges induced by I . G_1 and G_2 are said to be *simultaneously representable \mathcal{C} graphs* or *simultaneous \mathcal{C} graphs* if there exist intersection representations R_1 and R_2 of G_1 and G_2 such that any vertex of I is represented by the same object in both R_1 and R_2 . The *simultaneous representation problem* for class \mathcal{C} asks whether G_1 and G_2 are simultaneous \mathcal{C} graphs. For example, Figures 1(a) and 1(b) show two simultaneous interval graphs and their interval representations with the property that vertices common to both graphs are assigned to the same interval. Figure 1(c) shows two interval graphs that are not simultaneous interval graphs.

Simultaneous representation problems were introduced by us in a recent paper [9] and have application in any situation where two related graphs should be represented consistently. A main instance is for temporal relationships, where an old graph and a new graph share some common parts. Pairs of related graphs also arise in many other situations, e.g: two social networks that share some members; overlap graphs of DNA fragments of two similar organisms, etc.

*David R. Cheriton School of Computer Science, University of Waterloo, Email:krjampan@uwaterloo.ca

†David R. Cheriton School of Computer Science, University of Waterloo, Email:alubiw@uwaterloo.ca

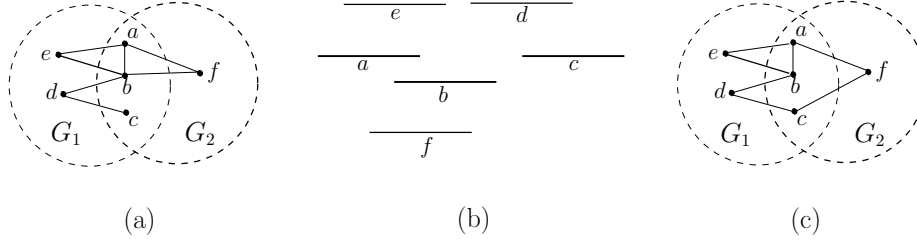


Figure 1: Graphs in (a) are simultaneous interval graphs as shown by the representations in (b). Graphs in (c) are not simultaneous interval graphs.

Simultaneous representations are related to simultaneous planar embeddings: two graphs that share some vertices and edges (not necessarily induced) have a *simultaneous geometric embedding* [3] if they have planar straight-line drawings in which the common vertices are represented by common points. Thus edges may cross, but only if they are in different graphs. Deciding if two graphs have a simultaneous geometric embedding is NP-Hard [4].

In [9], we showed that the simultaneous representation problem can be solved efficiently for chordal, comparability and permutation graphs. We also showed that for any intersection class \mathcal{C} , the simultaneous representation problem for G_1 and G_2 is equivalent to the following problem: Do there exist edges E' between $G_1 - I$ and $G_2 - I$ so that the augmented graph $G_1 \cup G_2 \cup E'$ belongs to class \mathcal{C} .

The *graph sandwich problem* [7] is a more general augmentation problem defined for any graph class \mathcal{C} : given graphs $H_1 = (V, E_1)$ and $H_2 = (V, E_2)$, is there a set E of edges with $E_1 \subseteq E \subseteq E_2$ so that the graph $G = (V, E)$ belongs to class \mathcal{C} . This problem has a wealth of applications but is NP-complete for interval graphs, comparability graphs, and permutation graphs [7].

The simultaneous representation problem (for class \mathcal{C}) is the special case of the graph sandwich problem (for \mathcal{C}) where $E_2 - E_1$ forms a complete bipartite subgraph. A related special case where $E_2 - E_1$ forms a clique is the problem of recognizing *probe graphs*: a graph G with a specified independent set N is a *probe graph* for class \mathcal{C} if there exist edges $E' \subseteq N \times N$ so that the augmented graph $G \cup E'$ belongs to class \mathcal{C} .

Probe graphs have several applications [13, 8] and have received much attention recently. The first polynomial-time algorithm for recognizing probe interval graphs was due to Johnson and Spinrad [10]. They used a variant of PQ-trees and achieved a run-time of $O(n^2)$. Techniques from modular decomposition provided more speed up [12], but the most recent algorithm by McConnell and Nussbaum [11] reverts to PQ-trees and achieves linear time.

We note that there has also been work [14] on a concept of simultaneous intersection called “polysemy” where two graphs are represented as intersections of sets and their complements.

In this paper, we give an $O(n^2 \log n)$ algorithm for solving the simultaneous representation problem for interval graphs. We use PQ-trees, which were developed by Booth and Lueker for the original linear time interval graph recognition algorithm. They used a PQ-tree to capture the orderings of the maximal cliques of the graph (see [6] for an introduction to interval graphs and PQ-trees).

In the probe interval recognition problem, there is a single PQ-tree (of the graph induced by the probes) and a set of constraints imposed by the non-probes. However in our situation we have two PQ-trees, one for each graph, that we want to re-order to “match” on the common vertex set I . We begin by “reducing” each PQ-tree to contain only vertices from I . This results in PQ-trees that store non-maximal cliques, and our task is to modify each PQ-tree by inserting non-maximal cliques from the other tree while re-ordering the trees to make them the same.

2 Reduction to PQ-trees

In this section we transform the simultaneous interval graph problem to a problem about “compatibility” of two PQ-trees arising from the two graphs.

Recall that an interval graph is defined to be the intersection graph of intervals on the real line. For any point on the line, the intervals containing that point form a clique in the graph. This leads to the fundamental one-to-one correspondence between the interval representations of an interval graph and its *clique orderings*, defined as follows: A *clique ordering* of G is a sequence of (possibly empty) cliques $\mathcal{S} = Q_1, Q_2, \dots, Q_l$ that contains all the maximal cliques of G and has the property that for each vertex v , the cliques in \mathcal{S} that contain v appear consecutively. Note that we allow cliques to be empty.

The standard interval graph recognition algorithm attempts to find a clique order of the maximal cliques of a graph by making the maximal cliques into leaves of a PQ-tree, and imposing PQ-tree constraints to ensure that the cliques containing each vertex v appear consecutively. This structure is called *the* PQ-tree of the graph. Note that the children of a P-node may be reordered arbitrarily and the children of a Q-node may only be reversed. We consider a node with 2 children to be a Q-node. In the figures, we use a circle to denote a P-node and a rectangle to denote a Q-node. A *leaf-order* of a PQ-tree is the order in which its leaves are visited in an in-order traversal of the tree, after children of P and Q-nodes are re-ordered as just described.

Note that ignoring non-maximal cliques is fine for recognizing interval graphs; for our purposes, however, we want to consider clique orders and PQ-trees that may include non-maximal cliques. We say that a PQ-tree whose leaves correspond to cliques of a graph is *valid* if for each of its leaf orderings and for each vertex v , the cliques containing v appear consecutively.

Let $\mathcal{S} = Q_1, Q_2, \dots, Q_l$ be a clique ordering of interval graph G and let the maximal cliques of G be $Q_{i_1}, Q_{i_2}, \dots, Q_{i_m}$ (appearing in positions $i_1 < i_2 < \dots < i_m$ respectively). Note that all the cliques in \mathcal{S} between Q_{i_j} and $Q_{i_{j+1}}$ contain $B = Q_{i_j} \cap Q_{i_{j+1}}$. We say that B is the *boundary clique* or *boundary* between Q_{i_j} and $Q_{i_{j+1}}$. Note that B may not necessarily be present in \mathcal{S} . The sequence of cliques between Q_{i_j} and $Q_{i_{j+1}}$ that are subsets of Q_{i_j} is said to be the *right tail* of Q_{i_j} . The *left tail* of $Q_{i_{j+1}}$ is defined analogously. Observe that the left tail of a clique forms an increasing sequence and the right tail forms a decreasing sequence (w.r.t set inclusion). Also note that all the cliques that precede Q_{i_1} are subsets of Q_{i_1} and this sequence is called the left tail of Q_{i_1} and all the cliques that succeed Q_{i_m} are subsets of Q_{i_m} and this sequence is called the right tail of Q_{i_m} . Thus any clique ordering of G consists of a sequence of maximal cliques, with each maximal clique containing a (possibly empty) left and right tail of subcliques.

Let Q_0 and Q_{l+1} be defined to be empty sets. An insertion of clique Q' between Q_i and Q_{i+1} (for some $i \in \{0, \dots, l\}$) is said to be a *subclique insertion* if $Q' \supseteq Q_i \cap Q_{i+1}$ and either $Q' \subseteq Q_i$ or $Q' \subseteq Q_{i+1}$. It is clear that after a subclique insertion the resulting sequence is still a clique ordering of G . A clique ordering \mathcal{S}' is an *extension* of \mathcal{S} if \mathcal{S}' can be obtained from \mathcal{S} by subclique insertions. We also say that \mathcal{S} extends to \mathcal{S}' . Furthermore, we say that a clique ordering is *generated* by a PQ-tree, if it can be obtained from a leaf order of the PQ-tree with subclique insertions. The above definitions yield the following Lemma.

Lemma 1. *A sequence of cliques \mathcal{S} is a clique ordering of G if and only if \mathcal{S} can be generated from the PQ-tree of G .*

Let G_1 and G_2 be two interval graphs sharing a vertex set I (i.e. $I = V(G_1) \cap V(G_2)$) and its induced edges. Note that $G_1[I]$ is isomorphic to $G_2[I]$. A clique ordering of $G_1[I]$ is said to be an I -ordering.

The I -restricted PQ-tree of G_j is defined to be the tree obtained from the PQ-tree of G_j by replacing each clique Q (a leaf of the PQ-tree) with the clique $Q \cap I$. Thus there is a one-to-one correspondence between the two PQ-trees, and the leaves of the I -restricted PQ-tree are cliques of $G_1[I]$.

Let $\mathcal{I} = X_1, X_2, \dots, X_l$ be an I -ordering. \mathcal{I} is said to be G_j -expandable if there exists a clique ordering $\mathcal{O} = Q_1, Q_2, \dots, Q_l$ of G_j such that $X_i \subseteq Q_i$ for $i \in \{1, \dots, l\}$. Further, we say that \mathcal{I} expands to \mathcal{O} . By the definition of clique-ordering it follows that, if \mathcal{I} is G_j -expandable then it remains G_j -expandable after a subclique insertion (i.e. any extension of \mathcal{I} is also G_j -expandable). We first observe the following.

Lemma 2. *The set of G_j -expandable I -orderings is same as the set of orderings that can be generated from the I -restricted PQ-tree of G_j .*

Proof. Let T be the PQ-tree of G_j and T' be the I -restricted PQ-tree of G_j .

Let \mathcal{I} be a G_j -expandable I -ordering of G_j . Then there exists a clique ordering \mathcal{O} of G_j such that \mathcal{I} expands to \mathcal{O} . But by Lemma 1, \mathcal{O} can be generated from T (from a leaf order with subclique insertions). This in turn implies that \mathcal{I} can be generated from T' (from the corresponding leaf order with the corresponding subclique insertions).

Now for the other direction, let $\mathcal{I}' = X_1, \dots, X_l$ be any leaf order of T' . Then there exists a corresponding leaf order $\mathcal{O}' = Q_1, \dots, Q_l$ of T such that $X_i \subseteq Q_i$ for $i \in \{1, \dots, l\}$. This implies that \mathcal{I}' is a G_j -expandable I -ordering. Finally, observe that if \mathcal{I}'' is generated from \mathcal{I}' by subclique insertions then \mathcal{I}'' is also a G_j -expandable I -ordering. Thus the Lemma holds. \square

Two I -orderings \mathcal{I}_1 and \mathcal{I}_2 are said to be *compatible* if both \mathcal{I}_1 and \mathcal{I}_2 (separately) extend to a common I -ordering \mathcal{I} . For e.g. the ordering $\{1\}, \{1, 2\}, \{1, 2, 3, 4\}$ is compatible with the ordering $\{1\}, \{1, 2, 3\}, \{1, 2, 3, 4\}$, as they both extend to the common ordering: $\{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 2, 3, 4\}$. Note that the compatibility relation is not transitive. Two PQ-trees T_1 and T_2 are said to be *compatible* if there exist orderings \mathcal{O}_1 and \mathcal{O}_2 generated from T_1 and T_2 (respectively) such that \mathcal{O}_1 is compatible with \mathcal{O}_2 . The following Lemma is our main tool.

Lemma 3. *G_1 and G_2 are simultaneous interval graphs if and only if the I -restricted PQ-tree of G_1 is compatible with the I -restricted PQ-tree of G_2 .*

Proof. By Lemma 2, it is enough to show that G_1 and G_2 are simultaneous interval graphs if and only if there exists a G_1 -expandable I -ordering \mathcal{I}_1 and a G_2 -expandable I -ordering \mathcal{I}_2 such that \mathcal{I}_1 is compatible with \mathcal{I}_2 . We now show this claim.

Let \mathcal{I}_1 and \mathcal{I}_2 be as defined in the hypothesis. Since \mathcal{I}_1 and \mathcal{I}_2 are compatible, they can be extended to a common I -ordering \mathcal{I} . Let \mathcal{I} expand to clique orderings \mathcal{O}_1 and \mathcal{O}_2 in G_1 and G_2 respectively. Since each vertex of I appears in the same positions in both \mathcal{O}_1 and \mathcal{O}_2 , it is possible to obtain interval representations R_1 and R_2 of G_1 and G_2 (from \mathcal{O}_1 and \mathcal{O}_2 respectively) such that each vertex in I has the same end points in both R_1 and R_2 . This implies that G_1 and G_2 are simultaneous interval graphs.

For the other direction, let G_1 and G_2 be simultaneous interval graphs. Then there exists an augmenting set of edges $A' \subseteq V_1 - I \times V_2 - I$ such that $G = G_1 \cup G_2 \cup A'$ is an interval graph. Let $\mathcal{O} = Q_1, Q_2, \dots, Q_l$ be a clique-ordering of G . For each $i \in \{1, \dots, l\}$ and $j \in \{1, 2\}$, by restricting Q_i to V_j (i.e. replacing Q_i with $Q_i \cap V_j$), we obtain a clique ordering \mathcal{O}_j of G_j . Now for $j \in 1, 2$, let \mathcal{I}_j be the I -ordering obtained from \mathcal{O}_j by restricting each clique in \mathcal{O}_j to I . It follows that \mathcal{I}_1 is a G_1 -expandable I -ordering and \mathcal{I}_2 is a G_2 -expandable I -ordering. Further $\mathcal{I}_1 = \mathcal{I}_2$ and hence \mathcal{I}_1 and \mathcal{I}_2 are compatible. \square

Our algorithm will decide if the I -restricted PQ-tree of G_1 is compatible with the I -restricted PQ-tree of G_2 . We first show how the I -restricted PQ-trees can be simplified in several ways. Two I -orderings \mathcal{I}_1 and \mathcal{I}_2 are said to be *equivalent* if for any I -ordering \mathcal{I}' , \mathcal{I}_1 and \mathcal{I}' are compatible if and only if \mathcal{I}_2 and \mathcal{I}' are compatible. Note that this is an equivalence relation. The Lemma below follows directly from the definitions of equivalent orderings and subclique insertions.

Lemma 4. Let $\mathcal{I} = X_1, X_2, \dots, X_l$ be an I -ordering in which $X_i = X_{i+1}$ for some $i \in 1, \dots, l-1$. Let \mathcal{I}' be the I -ordering obtained from \mathcal{I} by deleting X_{i+1} . Then \mathcal{I} is equivalent to \mathcal{I}' .

Further, because equivalence is transitive, Lemma 4 implies that an I -ordering \mathcal{I} is equivalent to the I -ordering \mathcal{I}' in which all consecutive duplicates are eliminated. This allows us to simplify the I -restricted PQ-tree of G_j . Let T be the I -restricted PQ-tree of G_j . We obtain a PQ-tree T' from T as follows.

1. Initialize $T' = T$.
2. As long as there is a non-leaf node n in T' such that all the descendants of n are the same, i.e. they are all duplicates of a single clique X , replace n and the subtree rooted at n by a leaf node representing X .
3. As long as there is a (non-leaf) Q-node n in T' with two consecutive child nodes n_a and n_b (among others) such that all the descendants of n_a and n_b are the same i.e. they are all duplicates of a single clique X , replace n_a, n_b and the subtrees rooted at these vertices by a single leaf node representing the clique X .

Note that the resulting T' is unique. We call T' the I -reduced PQ-tree of G_j .

Lemma 5. G_1 and G_2 are simultaneous interval graphs if and only if the I -reduced PQ-tree of G_1 is compatible with the I -reduced PQ-tree of G_2 .

Proof. For $j \in \{1, 2\}$, let T_j and T'_j be the I -restricted and I -reduced PQ-trees of G_j respectively. Let \mathcal{I} be any I -ordering. Observe that by Lemma 4, \mathcal{I} is compatible with a leaf ordering of T_j if and only if \mathcal{I} is compatible with a leaf ordering of T'_j . Thus the conclusion follows from Lemma 3. \square

3 Labeling and Further Simplification

In section 2, we transformed the simultaneous interval graph problem to a problem of testing compatibility of two I -reduced PQ-trees where I is the common vertex set of the two graphs. These PQ-trees may have nodes that correspond to non-maximal cliques in I . In this section we prove some basic properties of such I -reduced PQ-trees, and use them to further simplify each tree.

Let \mathcal{T} be the I -reduced PQ-tree of G_j . Recall that each leaf l of \mathcal{T} corresponds to a clique X in $G_j[I]$. If X is maximal in I , then X is said to be a *max-clique* and l is said to be a *max-clique node*, otherwise X is said to be a *subclique* and l is said to be a *subclique node*. When the association is clear from the context, we will sometimes refer to a leaf l and its corresponding clique X interchangeably, or interchange the terms “max-clique” and “max-clique node” [resp. subclique and subclique node]. A node of \mathcal{T} is said to be an *essential node* if it is a non-leaf node or if it is a leaf node representing a max-clique.

Given a node n of \mathcal{T} , the *descendant cliques* of n are the set of cliques that correspond to the leaf-descendants of n . Because our algorithm operates by inserting subcliques from one tree into the other, we must take care to preserve the validity of a PQ-tree. For this we need to re-structure the tree when we do subclique insertions. The required restructuring will be determined based on the label $U(n)$ that we assign to each node n as follows.

$U(n)$ or the *Universal set* of n is defined as the set of vertices v such that v appears in all descendant cliques of n .

Note that for a leaf node l representing a clique X , $U(l) = X$ by definition. Also note that along any path up the tree, the universal sets decrease. The following Lemma gives some useful properties of the I -reduced PQ-tree.

Lemma 6. *Let \mathcal{T} be the I -reduced PQ-tree of G_j . Let n be a non-leaf node of \mathcal{T} (n is used in properties 2–6). Then we have:*

- 0. *Let l_1 and l_2 be two distinct leaf nodes of \mathcal{T} , containing a vertex $t \in I$. Let y be the least common ancestor of l_1 and l_2 . Then: (a) If y is a P-node then all of its descendant cliques contain t . (b) If y is a Q-node then t is contained in all the descendant cliques of all children of y between (and including) the child of y that is the ancestor of l_1 and the child that is the ancestor of l_2 .*
- 1. *Each max-clique is represented by a unique node of \mathcal{T} .*
- 2. *A vertex u is in $U(n)$ if and only if for every child n_1 of n , $u \in U(n_1)$.*
- 3. *n contains a max-clique as a descendant.*
- 4. *If n is a P-node, then for any two child nodes n_1 and n_2 of n , we have $U(n) = U(n_1) \cap U(n_2)$.*
- 5. *If n is a P-node, then any child of n that is a subclique node represents the clique $U(n)$.*
- 6. *If n is a Q-node and n_1 and n_2 are the first and last child nodes of n then $U(n) = U(n_1) \cap U(n_2)$.*

Proof. (0) Observe that in any leaf ordering of \mathcal{T} , all the nodes that appear between l_1 and l_2 must also contain the vertex t , otherwise \mathcal{T} would be invalid. Now let l_3 be a leaf descendant of y , that doesn't contain t .

If y is a P-node, then we can reorder the children of y in such a way that in the leaf-ordering of the resulting tree l_3 appears between l_1 and l_2 . But this contradicts the validity of \mathcal{T} . This proves (a). Similarly, if y is a Q-node, then l_3 cannot be equal to l_1 or l_2 or any node between them. Thus (b) also holds.

(1) Note that by definition of I -reduced PQ-tree of G_j , each max-clique must be present in \mathcal{T} . Now assume for the sake of contradiction that a max-clique X is represented by two leaf nodes, say l_1 and l_2 . Let y be the least common ancestor of l_1 and l_2 . Let c_1 and c_2 be the child nodes of y that contain n_1 and n_2 (respectively) as descendants. Now by (0), if y is a P-node then all of its descendant cliques must contain all the vertices of X . But as X is maximal, all these cliques must be precisely X . However this is not possible, as we would have replaced y with a leaf node representing X in the construction of \mathcal{T} . Similarly, if y is a Q-node then the descendant cliques of c_1 , c_2 and all the nodes between them must represent the max-clique X . But then we would have replaced these nodes with a leaf node representing X in the construction of \mathcal{T} . This proves (1).

(2) If $u \in U(n)$, then all the descendant cliques of n contain u . This implies that for any child n_1 of n , all the descendant cliques of n_1 contain u . Hence $u \in U(n_1)$. On the other hand, if each child n_1 of n contains a vertex u in its universal set, then u is present in all the descendant cliques of n and thus $u \in U(n)$.

(3) Note that if each descendant clique of n contains precisely $U(n)$ (and no other vertex), then we would have replaced the subtree rooted at n with a leaf node corresponding to the clique $U(n)$, when constructing \mathcal{T} . Thus there exists a clique Q_2 that is a descendant of n , such that $Q_2 - U(n)$ is non-empty. If Q_2 is a max-clique then we are done. Otherwise let $t \in Q_2 - U(n)$ and let Q_1 be a max-clique containing t . Suppose Q_1 is not a descendant of n_1 . Applying (0) on Q_1 and Q_2 , we infer that irrespective of whether n is a P-node or a Q-node, all the descendant cliques of n must contain t . But then $t \in U(n)$, a contradiction. Thus Q_1 is a descendant of n_1 .

(4) By (2) we observe that $U(n) \subseteq U(n_1) \cap U(n_2)$. Thus it is enough to show that $U(n_1) \cap U(n_2) \subseteq U(n)$. Let $u \in U(n_1) \cap U(n_2)$, then u is present in all the descendant cliques of n_1 and n_2 . By (0), u must be present in all the descendant cliques of n and hence $u \in U(n)$. Therefore $U(n_1) \cap U(n_2) \subseteq U(n)$.

(5) Consider any child n_1 of n . Suppose n_1 is a leaf-node and is not a max-clique. It is enough to show that n_1 represents the clique $U(n)$ i.e. $U(n_1) = U(n)$. Suppose not. Then there exists a vertex $t \in U(n_1) - U(n)$. Let Q_1 be a max-clique containing t . Note that the common ancestor of Q_1 and n_1 is either n or an ancestor of n . Applying (0) on Q_1 and n_1 , we infer that all the descendant cliques of n must contain t . But then $t \in U(n)$, a contradiction.

(6) This follows from (2) and (0). □

Let \mathcal{T} be the I -reduced PQ-tree of G_j . Recall that an essential node is a non-leaf node or a leaf node representing a maximal clique. Equivalently (by Lemma 6.3), an essential node is a node which contains a max-clique as a descendant. The following Lemma shows that in some situations we can obtain an equivalent tree by deleting subclique child nodes of a P-node n . Recall that by Lemma 6.5, such subclique nodes represent the clique $U(n)$.

Lemma 7. *Let \mathcal{T} be the I -reduced PQ-tree of G_j and n be a P-node in \mathcal{T} . Then*

1. *If n has at least two essential child nodes, then \mathcal{T} is equivalent to the tree \mathcal{T}' , obtained from \mathcal{T} by deleting all the subclique children of n .*
2. *If n has at least two subclique child nodes, then \mathcal{T} is equivalent to the tree \mathcal{T}' , obtained from \mathcal{T} by deleting all except one of the subclique children of n .*

Proof. We give the proof of (1) below. The proof of (2) is very similar and hence omitted.

Let \mathcal{O}_1 be any I -ordering. It is enough to show that there exists a leaf ordering \mathcal{O} of \mathcal{T} that is compatible with \mathcal{O}_1 if and only if there exists a leaf ordering \mathcal{O}' of \mathcal{T}' that is compatible with \mathcal{O}_1 .

Let \mathcal{O} be any leaf ordering of \mathcal{T} , compatible with \mathcal{O}_1 . Consider the ordering \mathcal{O}' obtained from \mathcal{O} by deleting the cliques $U(n)$ that correspond to the child nodes of n in \mathcal{T} . Clearly \mathcal{O}' is a leaf ordering of \mathcal{T}' . Further \mathcal{O}' can be extended to \mathcal{O} by adding copies of the cliques $U(n)$ at appropriate positions. Thus \mathcal{O}' is compatible with \mathcal{O}_1 .

Now for the other direction, let \mathcal{O}' be a leaf order of \mathcal{T}' , compatible with \mathcal{O}_1 and let \mathcal{O}' and \mathcal{O}_1 extend to a common ordering \mathcal{O}_F . From the hypothesis, we can assume that there exist two essential child nodes n_1 and n_2 of n in \mathcal{T}' such that the clique descendants of n_1 , appear immediately before the clique descendants of n_2 in \mathcal{O}' . Also let $S(n_1)$ and $S(n_2)$ be the two subsequences of \mathcal{O}' containing the clique descendants of n_1 and n_2 respectively. Since n_1 and n_2 are essential nodes, $S(n_1)$ and $S(n_2)$ each contain at least one max-clique. Let Q_1 be the last max-clique in $S(n_1)$ and Q_2 be the first max-clique in $S(n_2)$. By Lemma 6.0, $Q_1 \cap Q_2 = U(n_1) \cap U(n_2) = U(n)$. Since \mathcal{O}' is compatible with \mathcal{O}_1 , in each of the two orderings \mathcal{O}' and \mathcal{O}_1 , Q_2 occurs after Q_1 and no other max-clique appears between them. Further the same holds for \mathcal{O}_F (as it is an extension of \mathcal{O}'). Let k be the number of subclique children of n (that represent the clique $U(n)$). Then obtain a leaf ordering \mathcal{O} of \mathcal{T} , from \mathcal{O}' , by inserting k copies of $U(n)$ between $S(n_1)$ and $S(n_2)$. Now extend \mathcal{O}_F to \mathcal{O}'_F by inserting k copies of $U(n)$ between Q_1 and Q_2 (there is a unique way of adding a subclique between two max-cliques). It is clear that \mathcal{O}'_F is an extension of both \mathcal{O} and \mathcal{O}_1 . Therefore \mathcal{O} is compatible with \mathcal{O}_1 . This proves (1). □

We will simplify \mathcal{T} as much as possible by applying Lemma 7 and by converting nodes with two children into Q-nodes. We call the end result a *simplified* I -reduced PQ-tree, but continue to use the term “ I -reduced PQ-tree” to refer to it. Note that the simplification process does not change the universal sets and preserves the validity of the PQ-tree so Lemma 5 and all the properties given in Lemma 6 still hold. Because we consider nodes with 2 children as Q-nodes Lemma 7 implies:

Corollary 1. *In a [simplified] I -reduced PQ-tree, any P-node has at least 3 children, and all the children are essential nodes.*

4 Algorithm

For $k \in \{1, 2\}$, let \mathcal{T}_k be the [simplified] I -reduced PQ-tree of G_k . By Lemma 5, testing whether G_1 and G_2 are simultaneous interval graphs is equivalent to testing whether \mathcal{T}_1 and \mathcal{T}_2 are compatible. We test this by modifying \mathcal{T}_1 and \mathcal{T}_2 (e.g. inserting the sub-clique nodes from one tree into the other) so as to make them identical, without losing their compatibility. The following is a high level overview of our approach for checking whether \mathcal{T}_1 and \mathcal{T}_2 are compatible.

Our algorithm is iterative and tries to *match* essential nodes of \mathcal{T}_1 with essential nodes of \mathcal{T}_2 in a bottom-up fashion. An essential node n_1 of \mathcal{T}_1 is matched with an essential node n_2 of \mathcal{T}_2 if and only if the subtrees rooted at n_1 and n_2 are the same i.e. their essential children are matched, their subclique children are the same and furthermore (in the case of Q-nodes) their child nodes appear in the same order. If n_1 is matched with n_2 then we consider n_1 and n_2 to be identical and use the same name (say n_1) to refer to either of them. Initially, we match each max-clique node of \mathcal{T}_1 with the corresponding max-clique node of \mathcal{T}_2 . Note that every max-clique node appears uniquely in each tree by Lemma 6.1. A sub-clique node may appear in only one tree in which case we must first insert it into the other tree. This is done when we consider the parent of the subclique node.

In each iteration, we either match an unmatched node u of \mathcal{T}_1 to an unmatched node v of \mathcal{T}_2 (which may involve inserting subclique child nodes of v as child nodes of u and vice versa) or we *reduce* either \mathcal{T}_1 or \mathcal{T}_2 without losing their compatibility relationship. *Reducing* a PQ-tree means restricting it to reduce the number of leaf orderings. Finally, at the end of the algorithm either we have modified \mathcal{T}_1 and \mathcal{T}_2 to a “common” tree \mathcal{T}_I that establishes their compatibility or we conclude that \mathcal{T}_1 is not compatible with \mathcal{T}_2 . The common tree \mathcal{T}_I is said to be an *intersection tree* (of \mathcal{T}_1 and \mathcal{T}_2) and has the property that any ordering generated by \mathcal{T}_I can also be generated by \mathcal{T}_1 and \mathcal{T}_2 . If \mathcal{T}_1 and \mathcal{T}_2 are compatible, there may be several intersection trees of \mathcal{T}_1 and \mathcal{T}_2 , but our algorithm finds only one of them.

We need the following additional notation for the rest of this paper. A sequence of subcliques $\mathcal{S} = X_1, X_2, \dots, X_l$ is said to satisfy the *subset property* if $X_i \subseteq X_{i+1}$ for $i \in \{1, \dots, l-1\}$. \mathcal{S} is said to satisfy the *superset property* if $X_i \supseteq X_{i+1}$ for each i . Note that \mathcal{S} satisfies the subset property if and only if $\bar{\mathcal{S}} = X_l, \dots, X_2, X_1$ satisfies the superset property.

Let d be an essential child node of a Q-node in \mathcal{T}_k . We will overload the term “tail” (previously defined for a max clique in a clique ordering) and define the *tails* of d as follows. The left tail (resp. right tail) of d is defined as the sequence of subcliques that appear as siblings of d , to the immediate left (resp. right) of d , such that each subclique is a subset of $U(d)$. Note that the left tail of d should satisfy the subset property and the right tail of d should satisfy the superset property (otherwise \mathcal{T}_k will not be valid). Also note that since the children of a Q-node can be reversed in order, “left” and “right” are relative to the child ordering of the Q-node. We will be careful to use “left tail” and “right tail” in such a way that this ambiguity does not matter. Now suppose d is a matched node. Then in order to match the parent of d in \mathcal{T}_1 with the parent of d in \mathcal{T}_2 , our algorithm has to “merge” the tails of d .

Let \mathcal{L}_1 and \mathcal{L}_2 be two subclique sequences that satisfy the subset property. Then \mathcal{L}_1 is said to be *mergable* with \mathcal{L}_2 if the union of subcliques in \mathcal{L}_1 and \mathcal{L}_2 can be arranged into an ordering \mathcal{L}' that satisfies the subset property. Analogously, if \mathcal{L}_1 and \mathcal{L}_2 satisfy the superset property, then they are said to be *mergable* if the union of their subcliques can be arranged into an ordering \mathcal{L}' that satisfies the superset property. In both cases, \mathcal{L}' is said to be the *merge* of \mathcal{L}_1 and \mathcal{L}_2 and is denoted by $\mathcal{L}_1 + \mathcal{L}_2$.

A *maximal matched node* is a node that is matched but whose parent is not matched. For an unmatched essential node x , the *MM-descendants* of x , denoted by $MMD(x)$ are its descendants that are maximal matched nodes. If x is matched then we define $MMD(x)$ to be the singleton set containing x . Note that the MM-descendants of an essential node is non-empty (since every essential node has a max-clique

descendant).

Our algorithm matches nodes from the leaves up, and starts by matching the leaves that are max-cliques. As the next node n_1 that we try to match, we want an unmatched node whose essential children are already matched. To help us choose between \mathcal{T}_1 and \mathcal{T}_2 , and also to break ties, we prefer a node with larger U set. Then, as a candidate to match n_1 to, we want an unmatched node in the other tree that has some matched children in common with n_1 . With this intuition in mind, our specific rule is as follows.

Among all the unmatched essential nodes of \mathcal{T}_1 union \mathcal{T}_2 choose n_1 with maximal $U(n_1)$, minimal $MMD(n_1)$, and maximal depth, in that preference order. Assume without loss of generality that $n_1 \in \mathcal{T}_1$. Select an unmatched node n_2 from \mathcal{T}_2 with maximal $U(n_2)$, minimal $MMD(n_2)$ and maximal depth (in that order) satisfying the property that $MMD(n_1) \cap MMD(n_2) \neq \emptyset$. The following Lemma captures certain properties of n_1 and n_2 , including why these rules match our intuitive justification.

Lemma 8. *For n_1 and n_2 chosen as described above, let $M_1 = MMD(n_1)$, $M_2 = MMD(n_2)$ and $X = M_1 \cap M_2$. Also let C_1 and C_2 be the essential child nodes of n_1 and n_2 respectively. Then we have:*

1. $M_1 = C_1$ and $X \subseteq C_2$.

Further when \mathcal{T}_1 is compatible with \mathcal{T}_2 , we have:

2. *For every (matched) node l in $M_1 - X$ of \mathcal{T}_1 , its corresponding matched node l' in \mathcal{T}_2 is present outside the subtree rooted at n_2 . Analogously, for every (matched) node r' in $M_2 - X$ of \mathcal{T}_2 , its corresponding matched node r in \mathcal{T}_1 is present outside the subtree rooted at n_1 .*

3. *If n_1 [resp. n_2] is a Q-node, then in its child ordering, no node of $C_1 - X$ [resp. $C_2 - X$] can be present between two nodes of X .*

4. *If n_1 and n_2 are Q-nodes, then in the child ordering of n_1 and n_2 , nodes of X appear in the same relative order i.e. for any three nodes $x_1, x_2, x_3 \in X$, x_1 appears between x_2 and x_3 in the child ordering of n_1 if and only if x_1 also appears between x_2 and x_3 in the child ordering of n_2 .*

5. *If $C_1 - X$ (resp. $C_2 - X$) is non-empty then $U(n_1) \subseteq U(n_2)$ (resp. $U(n_2) \subseteq U(n_1)$). Further, if $C_1 - X$ is non-empty then so is $C_2 - X$ and hence $U(n_1) = U(n_2)$.*

6. *Let $C_1 - X$ be non-empty. If n_1 [resp. n_2] is a Q-node, then in its child-ordering either all nodes of $C_1 - X$ [resp. $C_2 - X$] appear before the nodes of X or they all appear after the nodes of X .*

Proof. (1) If there exists an unmatched child c of n_1 , then as $U(c) \supseteq U(n_1)$, $MMD(c) \subseteq MMD(n_1)$ and c has a greater depth than n_1 , we would have chosen c over n_1 . Thus every node in C_1 is matched and hence by the definition of MM-descendants $C_1 = M_1$.

For the second part, suppose there exists a node $x \in X$ that is not a child of n_2 . Let c_2 be the child of n_2 that contains x as a descendant. c_2 must be an unmatched node. (Otherwise $MMD(n_2)$ would have contained c_2 and not x). But then we would have picked c_2 over n_2 .

(2) Let l be a (matched) node in $M_1 - X$ (in \mathcal{T}_1) such that the corresponding matched node l' in \mathcal{T}_2 is a descendant of n_2 . Note that l' cannot be a child of n_2 . Otherwise $l' \in M_2$ and thus $l = l'$ is in X . Let p' be the parent of l' . Now p' cannot be a matched node. (Otherwise p' would have been matched to n_1 , a contradiction that n_1 is unmatched). Also p' is a descendant of n_2 and hence $U(p') \supseteq U(n_2)$, $MMD(p') \subseteq MMD(n_2)$ and p' has greater depth than n_2 . Further $l = l'$ is a common MM-descendant of n_1 and p' . This contradicts the choice of n_2 .

Now let r' be a (matched) node in $M_2 - X$ (in \mathcal{T}_2), such that the corresponding matched node r is a descendant of n_1 . Note that r is not a child of n_1 , otherwise $r = r'$ is a common MM-descendant of n_1 and n_2 and hence $r' = r \in X$. Let p be the parent of r in \mathcal{T}_1 . Since p is a proper descendant of n_1 , p is a matched node. Let p be matched to a node p' in \mathcal{T}_2 . Now p' is a parent of r' and a descendant of n_2 . But then the MM-descendants of n_2 should not have contained r' .

(3) Suppose in the child ordering of n_1 , node $y \in C_1 - X$ is present between nodes $x_a \in X$ and $x_b \in X$. Let Y, X_a and X_b be any max-cliques that are descendants of y, x_a and x_b respectively. Then in any ordering of \mathcal{T}_1 , Y appears between X_a and X_b . But by (2), the corresponding matched node y' of y in \mathcal{T}_2 appears outside the subtree rooted at n_2 . Thus in any ordering of \mathcal{T}_2 , Y appears either before or after both X_a and X_b . Thus \mathcal{T}_1 and \mathcal{T}_2 are not compatible. This shows the claim for n_1 . The proof for n_2 is similar.

(4) This follows from the fact that \mathcal{T}_1 and \mathcal{T}_2 are compatible and by observing that each matched node (in particular any node in X) contains a max-clique as a descendant.

(5) Let $x_a \in X$ be a common child of n_1 and n_2 . Let X_a be a max-clique descendant of x_a . Suppose $C_1 - X$ is non-empty. Then let Y_a be any max-clique descendant of a node in $C_1 - X$. Note that by (2), Y_a is present outside the subtree rooted at n_2 . Now by Lemma 6.0 and observing that the least common ancestor of X_a and Y_a (in \mathcal{T}_2) is an ancestor of n_2 , we get $U(n_2) \supseteq X_a \cap Y_a \supseteq U(n_1)$. Thus $U(n_1) \subseteq U(n_2)$. Using an analogous argument we can show that if $C_2 - X$ is non-empty then $U(n_2) \subseteq U(n_1)$. This proves the first part of the property.

For the second part we once again assume that $C_1 - X$ is non-empty and hence $U(n_1) \subseteq U(n_2)$. Now if $C_2 - X$ is empty then $MMD(n_2) = X \subset MMD(n_1)$. But this contradicts the choice of n_1 (we would have selected n_2 instead).

(6) By (5), $C_2 - X$ is non-empty and $U(n_1) = U(n_2)$. Let $x_a \in X$ and suppose $y_a, y_b \in C_1 - X$ are any two nodes on different sides of X . Let $z_a \in C_2 - X$. Note that by (2), the matched nodes of y_a, y_b in \mathcal{T}_2 appear outside the subtree rooted at n_2 and the matched node of z_a in \mathcal{T}_1 appears outside the subtree rooted at n_1 . Now let X_a, Y_a, Y_b and Z_a be any descendant max-cliques of x_a, y_a, y_b and z_a respectively. In any leaf-ordering of \mathcal{T}_1 , X_a appears between Y_a and Y_b , and Z_a doesn't appear between Y_a and Y_b . But in any leaf-ordering of \mathcal{T}_2 , either Z_a and X_a both appear between Y_a and Y_b or they both appear before or after Y_a and Y_b . This contradicts that \mathcal{T}_1 and \mathcal{T}_2 are compatible. Therefore all nodes of X appear before or after all nodes of $C_1 - X$ in the child ordering of n_1 . Similarly, the claim also holds for the child ordering of n_2 in \mathcal{T}_2 . □

We now describe the main step of the algorithm. Let $n_1, n_2, M_1, M_2, C_1, C_2$ and X be as defined in the above Lemma. We have four cases depending on whether n_1 and n_2 are P or Q-nodes. In each of these cases, we make progress by either matching two previously unmatched essential nodes of \mathcal{T}_1 and \mathcal{T}_2 or by reducing \mathcal{T}_1 and/or \mathcal{T}_2 at n_1 or n_2 while preserving their compatibility. We show that our algorithm requires at most $O(n \log n)$ iterations and each iteration takes $O(n)$ time. Thus our algorithm runs in $O(n^2 \log n)$ time.

During the course of the algorithm we may also insert subcliques into a Q-node when we are trying to match it to another Q-node. This is potentially dangerous as it may destroy the validity of the PQ-tree. When the Q-nodes have the same universal set, this trouble does not arise. However, in case the two Q-nodes have different universal sets, we need to re-structure the trees. Case 4, when n_1 and n_2 are both Q-nodes, has subcases to deal with these complications.

Case 1: n_1 and n_2 are both P-nodes.

By Corollary 1, the children of n_1 and n_2 are essential nodes, so C_1 and C_2 are precisely the children of n_1 and n_2 respectively. Let X consist of nodes $\{x_1, \dots, x_{k_0}\}$. If $C_2 - X$ is empty, then by Lemma 8.5, $C_1 - X$

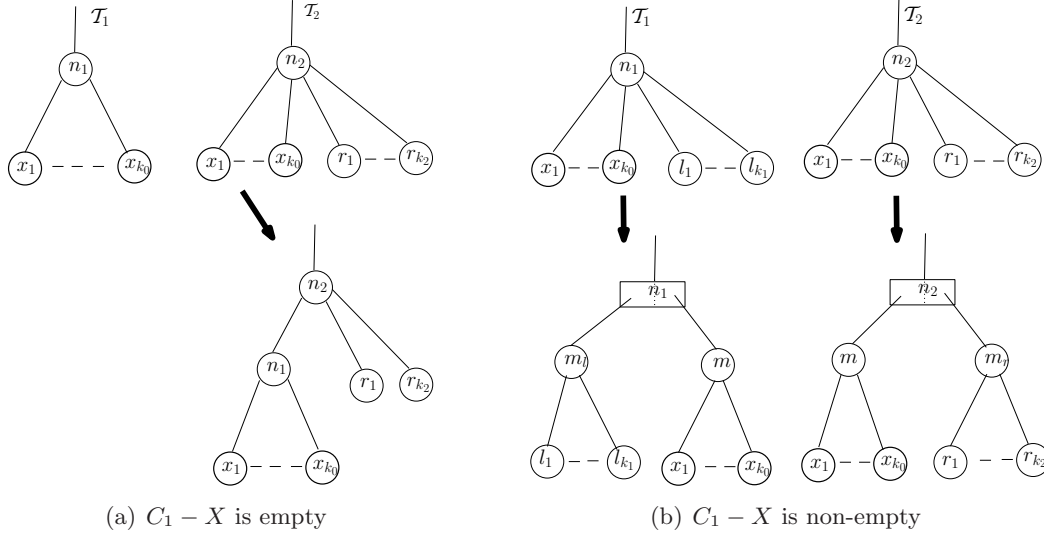


Figure 2: Reduction templates for Case 1

is also empty and hence n_1 and n_2 are the same. So we match n_1 with n_2 and go to the next iteration. Suppose now that $C_2 - X$ is non empty. Let $C_2 - X = \{r_1, \dots, r_{k_2}\}$. If $C_1 - X$ is empty, then we use the reduction template of Figure 2(a) to modify \mathcal{T}_2 , matching the new parent of X in \mathcal{T}_2 to n_1 . It is easy to see that \mathcal{T}_1 is compatible with \mathcal{T}_2 if and only if \mathcal{T}_1 is compatible with the modified \mathcal{T}_2 .

Now let $C_1 - X = \{l_1, \dots, l_{k_1}\}$ be non-empty. In this case we use the reduction template of Figure 2(b) to modify \mathcal{T}_1 and \mathcal{T}_2 to \mathcal{T}'_1 and \mathcal{T}'_2 respectively. Note that it is possible to have $k_i = 1$ for some i 's, in which case the template is slightly different because we do not make a node with one child, however, the reduction always makes progress as each n_i has at least 3 children.

We now claim that \mathcal{T}_1 is compatible with \mathcal{T}_2 if and only if \mathcal{T}'_1 is compatible with \mathcal{T}'_2 . The reverse direction is trivial. For the forward direction, let \mathcal{O}_1 and \mathcal{O}_2 be two compatible leaf orderings of \mathcal{T}_1 and \mathcal{T}_2 respectively. Recall that by Lemma 8.2, for every [matched] node of $C_1 - X$ in \mathcal{T}_1 , the corresponding matched node in \mathcal{T}_2 appears outside the subtree rooted at n_2 . This implies that the descendant nodes of $\{x_1, x_2, \dots, x_{k_0}\}$ all appear consecutively in \mathcal{O}_1 . Hence the descendant nodes of $\{x_1, x_2, \dots, x_{k_0}\}$ also appear consecutively in \mathcal{O}_2 . Thus we conclude that \mathcal{T}_1 and \mathcal{T}_2 are compatible if and only if the reduced trees \mathcal{T}'_1 and \mathcal{T}'_2 are also compatible. Note that both the template reductions take at most $O(n)$ time.

Case 2: n_1 is a P-node and n_2 is a Q-node.

If $C_1 - X = \emptyset$, we reduce \mathcal{T}_1 by ordering the children of n_1 as they appear in the child ordering of n_2 , and changing n_1 into a Q-node (and leading to Case 4). This reduction preserves the compatibility of the two trees.

Now suppose $C_1 - X \neq \emptyset$. Lemma 8.5 implies that, $C_2 - X \neq \emptyset$ and $U(n_1) = U(n_2)$. By Lemma 8.6, we can assume that the nodes in X appear before the nodes in $C_2 - X$ in the child ordering of n_2 . Now let $X = x_1, \dots, x_{k_0}$, $C_1 - X = l_1, \dots, l_{k_1}$ and $C_2 - X = r_1, \dots, r_{k_2}$. For $i \in 2, \dots, k_0$, let \mathcal{S}_i be the sequence of subcliques that appear between x_{i-1} and x_i in the child ordering of n_2 . Note that \mathcal{S}_i consists of the right tail of x_{i-1} followed by the left tail of x_i . We let \mathcal{S}_1 and \mathcal{S}_{k_0+1} denote the left and right tails of x_1 and x_{k_0} respectively. We now reduce the subtree rooted at n_1 as shown in Figure 3, changing it into a Q-node. Clearly $U(n_1)$ is preserved in this operation. The correctness of this operation follows by Lemma

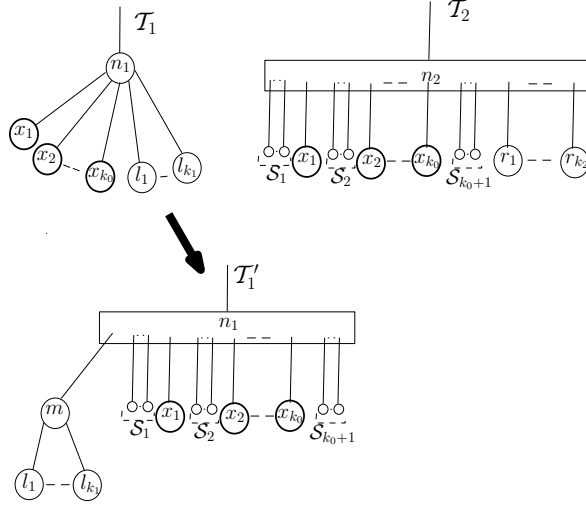


Figure 3: Reduction template for Case 2, when $C_1 - X \neq \emptyset$

8.2. It is easy to see that both the template reductions run in $O(n)$ time.

Case 3: n_1 is a Q-node and n_2 is a P-node.

If $C_2 - X$ is empty, then we reduce \mathcal{T}_2 by ordering the child nodes of n_2 (i.e. X) as they appear in the child ordering of n_1 , and changing n_2 into a Q-node.

Now let $C_2 - X$ be nonempty. By Lemma 8.5, $U(n_2) \subseteq U(n_1)$. Let $X = \{x_1, x_2, \dots, x_{k_0}\}$, $C_2 - X = \{r_1, \dots, r_{k_2}\}$ and $\mathcal{S}_1, \dots, \mathcal{S}_{k_0+1}$ be defined as in the previous case: \mathcal{S}_1 is the left tail of x_1 (in \mathcal{T}_1), \mathcal{S}_i is the concatenation of the right tail of x_{i-1} and the left tail of x_i , for $i \in \{2, \dots, k_0\}$ and \mathcal{S}_{k_0+1} is the right tail of x_{k_0} .

Now if $C_1 - X$ is empty, then we use the template of Figure 4 to reduce \mathcal{T}_2 , grouping all nodes of X into a new Q-node w , ordering them in the way they appear in \mathcal{T}_1 and inserting the subclique children of n_1 into w . Note that since $U(n_2) \subseteq U(n_1)$, this operation doesn't change $U(n_2)$ and hence it preserves the validity of \mathcal{T}_2 . Further n_1 is identical to w and hence we match these nodes. Thus we make progress even when $|X| = 1$.

If $C_1 - X$ is non-empty, we use the template similar to Figure 3 (to reduce \mathcal{T}_2) in which the roles of n_1 and n_2 have been switched. Note that the template reductions of this case run in $O(n)$ time.

Case 4: n_1 and n_2 are both Q-nodes

Let $X = \{x_1, \dots, x_{k_0}\}$ appear in that order in the child ordering of n_1 and n_2 . (They appear in the same order because of Lemma 8.4.) Let p_1 and p_2 be the parents of n_1 and n_2 respectively.

If n_1 and n_2 have no other children than X , we match n_1 with n_2 and proceed to the next iteration. More typically, they have other children. These may be essential nodes to one side or the other of X (by Lemma 8.6) or subclique nodes interspersed in X as tails of the nodes of X . We give a high-level outline of Case 4, beginning with a discussion of subclique nodes.

For $i \in \{1, \dots, k_0\}$, let \mathcal{L}_i and \mathcal{R}_i be the left and right tails of x_i in \mathcal{T}_1 and, \mathcal{L}'_i and \mathcal{R}'_i be the left and right tails of x_i in \mathcal{T}_2 . The only way to deal with the subclique nodes is to do subclique insertions in both trees to merge the tails. This is because in any intersection tree \mathcal{T}_I obtained from \mathcal{T}_1 and \mathcal{T}_2 , the tails of x_i in \mathcal{T}_I must contain the merge of the tails of x_i in \mathcal{T}_1 and \mathcal{T}_2 . So long as $|X| \geq 2$, the ordering x_1, \dots, x_{k_0}

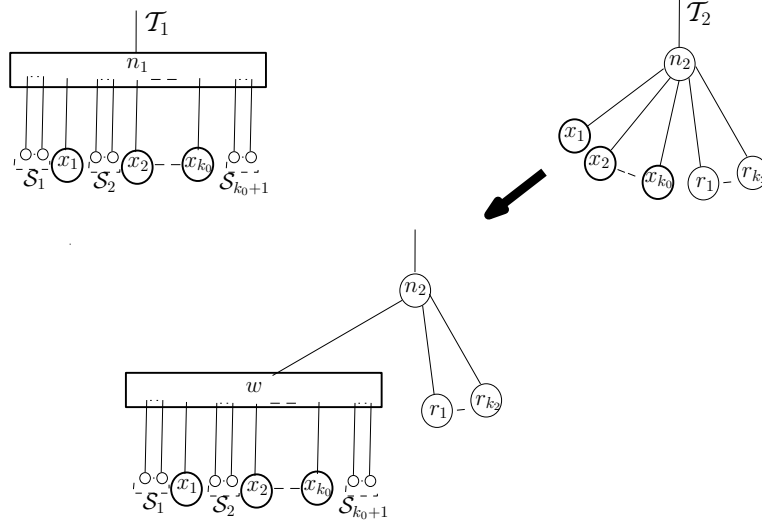


Figure 4: Reduction template for Case 3, when n_1 is a Q-node, n_2 is a P-node and $C_1 - X$ is empty.

completely determines which pairs of tails must merge: \mathcal{L}_i must merge with \mathcal{L}'_i and \mathcal{R}_i must merge with \mathcal{R}'_i .

The case $|X| = 1$ is more complicated because the intersection tree may merge \mathcal{L}_1 with \mathcal{L}'_1 and \mathcal{R}_1 with \mathcal{R}'_1 or merge \mathcal{L}_1 with $\bar{\mathcal{R}}_1$ and \mathcal{R}_1 with $\bar{\mathcal{L}}_1$. This decision problem is referred to as the *alignment problem*. We prove (at the beginning of Case 4.3) that in case both choices give mergable pairs, then either choice yields an intersection tree, if an intersection tree exists.

This completes our high-level discussion of subclique nodes. We continue with a high-level description of the subcase structure for Case 4. We have subcases depending on whether $U(n_1) = U(n_2)$ and whether n_1 and n_2 have the same essential children. If both these conditions hold, then we merge the tails of the nodes of X and match n_1 with n_2 . (In other words we replace \mathcal{L}_i and \mathcal{L}'_i with $\mathcal{L}_i + \mathcal{L}'_i$, and replace \mathcal{R}_i and \mathcal{R}'_i with $\mathcal{R}_i + \mathcal{R}'_i$). The cost of matching any two nodes x and y is $(m_x + m_y)|I|$, where m_x and m_y are the number of subclique children of x and y respectively. Once a node is matched its subclique children will not change. Hence the total amortized cost of matching all the nodes is $O(n \cdot |I|) = O(n^2)$.

When $U(n_1) \neq U(n_2)$ or when n_1 and n_2 do not have the same essential children then we have three subcases. Case 4.1 handles the situation when $U(n_1) \not\supseteq U(n_2)$. In this case we either insert subcliques of one tree into another and match n_1 with n_2 or we do some subclique insertions that will take us to the case when $U(n_1) \supseteq U(n_2)$. The remaining cases handle the situation when $U(n_1) \supseteq U(n_2)$, Case 4.2 when $C_1 - X$ is non-empty and Case 4.3 when it is empty. In both cases, we reduce \mathcal{T}_1 but the details vary. However in both cases our reduction templates depend on whether p_1 is a P-node or a Q-node. If p_1 is a P-node, we reduce \mathcal{T}_1 by grouping some of the child nodes of p_1 into a single node, deleting them and adding the node as a first or last child of n_1 . If p_1 is a Q-node then there are two ways of reducing: delete n_1 and reassign its children as children of p_1 or reverse the children of n_1 , delete n_1 and reassign its children as children of p_1 . We refer to this operation as a *collapse*. We now give the details of each case.

Case 4.1: $U(n_1) \not\supseteq U(n_2)$

Since n_1 was chosen so that $U(n_1)$ is maximal, we also have $U(n_2) \not\supseteq U(n_1)$. Now using Lemma 8.5, we infer that $C_1 - X$ is empty and $C_2 - X$ is empty. Thus the difference between $U(n_1)$ and $U(n_2)$ arises due

to the subcliques. Let L be a subclique that is either the first or the last child of n_1 , with the property that $L \not\subseteq U(n_2)$. Such a subclique exists since by Lemma 6.6, the intersection of the universal sets of the first and last child nodes of n_1 is $U(n_1)$. Also let R be a subclique that is either the first or the last child of n_2 , with the property that $R \not\subseteq U(n_1)$.

Note that even if $|X| = 1$, the alignment is unique since L and R cannot appear in the same tail of x_1 in any intersection tree. Further, we can assume without loss of generality that L is present in the left tail of x_1 in \mathcal{T}_1 and R is present in the right tail of x_{k_0} in \mathcal{T}_2 .

Let X_1 be any max-clique descendant of x_1 . If p_1 is a P-node then we claim that $U(p_1) \subseteq U(n_2)$. To see this, let Z be a max-clique descendant of p_1 , that is not a descendant of n_1 . In \mathcal{T}_2 , Z appears outside the subtree rooted at n_2 . Now by applying Lemma 6.0 on X_1 and Z , we conclude that every descendant of n_2 must contain the vertex set $Z \cap X_1$. Thus we have $U(p_1) \subseteq (Z \cap X_1) \subseteq U(n_2)$. Note that if \mathcal{T}_1 and \mathcal{T}_2 are compatible, then in any intersection tree of \mathcal{T}_1 and \mathcal{T}_2 , the nodes of \mathcal{L}_1 and \mathcal{L}'_1 appear in the left tail of x_1 and the nodes of \mathcal{R}_{k_0} and \mathcal{R}'_{k_0} appear in the right tail of x_{k_0} . Now if \mathcal{L}'_1 is non-empty, then we insert the left most subclique of \mathcal{L}'_1 into \mathcal{L}_1 (at the appropriate location so that the resulting sequence is still a subclique ordering), as a child of n_1 . Also if \mathcal{R}'_{k_0} is non-empty, then we insert the right most subclique of \mathcal{R}'_{k_0} into \mathcal{R}_{k_0} , as a child of n_1 . These insertions change $U(n_1)$ to $U(n_1) \cap U(n_2) \supseteq U(p_1)$ and we would be in case 4.3 with the roles of n_1 and n_2 being reversed. (Note that since the universal set of the modified n_1 is a superset of the universal set of p_1 , the resulting reduced tree of \mathcal{T}_1 is valid). Although this doesn't constitute a progress step since the number of leaf orderings of n_1 doesn't change, we will make progress in the Case 4.3.

Similarly if p_2 is a P-node then we insert the first subclique of \mathcal{L}_1 (if it exists) into \mathcal{L}'_1 and the last subclique of \mathcal{R}_{k_0} (if it exists) into \mathcal{R}'_{k_0} . After this we would be in Case 4.3.

Now if the parents of n_1 and n_2 are both Q-nodes then we look at the tails of n_1 and n_2 . If all the subcliques in these tails are subsets of $U(n_1) \cap U(n_2)$, then we replace \mathcal{L}_i and \mathcal{L}'_i with $\mathcal{L}_i + \mathcal{L}'_i$ and \mathcal{R}_i and \mathcal{R}'_i with $\mathcal{R}_i + \mathcal{R}'_i$. This changes $U(n_1)$ and $U(n_2)$ to $U(n_1) \cap U(n_2)$ and makes n_1 identical to n_2 . Thus we match n_1 with n_2 and iterate.

Otherwise without loss of generality let the subclique $S \not\subseteq U(n_2)$ be present in the (say left) tail of n_1 . Observe that in any intersection tree S and R cannot be present in the same tail of x_{k_0} (since neither is a subset of the other). This implies that we can reduce the tree \mathcal{T}_1 by collapsing n_1 i.e. by removing n_1 , inserting the sequence of child nodes of n_1 after S (S and L are now in the left tail of x_1), and assigning p_1 as their parent. This completes case 4.1. Note that all the steps in this case take $O(n)$ time, except the matching step (recall that all the matching steps take $O(n^2)$ amortized time).

Case 4.2: $U(n_1) \supseteq U(n_2)$ and $C_1 - X$ is non-empty.

By Lemma 8.5, $C_2 - X$ is also non-empty and further $U(n_1)$ is equal to $U(n_2)$. In this case we will reduce \mathcal{T}_1 depending on whether p_1 is a P-node or a Q-node. Further when p_1 is a Q-node, our reduction template also depends on whether n_1 has sibling essential nodes.

Let l_1, l_2, \dots, l_{k_1} be the essential nodes in $C_1 - X$ appearing in that order and appearing (without loss of generality) before the nodes of X in \mathcal{T}_1 . Note that by Lemma 8.2, for each node in $C_1 - X$, the corresponding matched node in \mathcal{T}_2 appears outside the subtree rooted at n_2 . Thus if \mathcal{T}_1 and \mathcal{T}_2 are compatible, then all the nodes of $C_2 - X$ must appear after the nodes of X in the child ordering of n_2 . Let these nodes be r_1, r_2, \dots, r_{k_2} .

Case 4.2.1: p_1 is a P-node

Let $Y = \{y_1, y_2, \dots, y_{k_3}\}$ be the child nodes of p_1 other than n_1 . Also, let \mathcal{T}_I be any intersection tree of \mathcal{T}_1 and \mathcal{T}_2 . We first observe that for $i \in \{1, \dots, k_0\}$ and $j \in \{1, \dots, k_2\}$, $MMD(y_i) \cap MMD(r_j) \neq \emptyset$, if

and only if y_i and r_j have a max-clique descendant.

For any such pair y_i and r_j , let $MMD(y_i) \cap MMD(r_j) \neq \emptyset$ and let Y be a common max-clique descendant of y_i and r_j . Then note that because of the constraints imposed by the child ordering of n_2 , in any leaf ordering of \mathcal{T}_I , the descendant cliques of l_1 do not appear between the descendant cliques of x_1 and Y . Thus y_i must appear after x_{k_1} , and so we reduce \mathcal{T}_1 , by grouping all nodes y_i satisfying $MMD(y_i) \cap MMD(r_j) \neq \emptyset$ for some r_j into a P-node and adding it as a child node of n_1 to the (immediate) right of \mathcal{R}_{k_0} as shown in Figure 5(top).

Now if $MMD(y_i) \cap MMD(r_j) = \emptyset$ for all y_i and r_j , then the above reduction doesn't apply. But in this case (because of the constraints on n_2), for every y_i and every leaf ordering of \mathcal{T}_I , no max-clique descendant of y_i appears between the max-clique descendants of n_2 . Thus we group all the nodes of Y into a P-node and add it as a child of n_1 to the left of l_1 as shown in Figure 5(bottom).

Note that for any two distinct nodes $y_a, y_b \in \{y_1, \dots, y_{k_3}\}$ we have: $MMD(y_a) \cap MMD(y_b) = \emptyset$. Similarly, for any two distinct nodes $r_a, r_b \in \{r_1, \dots, r_{k_2}\}$ we have: $MMD(r_a) \cap MMD(r_b) = \emptyset$. This implies that we can first compute the MM-Descendants of all y_i and r_j in $O(n)$ time and further we can compute all nodes y_i that satisfy $MMD(y_i) \cap MMD(r_j) \neq \emptyset$ for some r_j , in $O(n)$ time. Thus the template reductions of Figure 5 run in $O(n)$ time.

Case 4.2.2: p_1 is a Q-node and n_1 is its only essential child.

Since the only essential child of p_1 is n_1 , all of its remaining children are subcliques that are present as tails of n_1 . Thus each of these subcliques is a subset of $U(n_1)$. Now let Z and R be any two max-clique descendants of x_{k_0} and r_1 respectively. By Lemma 8.2, R appears outside the subtree rooted at n_1 (in \mathcal{T}_1) and hence outside the subtree rooted at p_1 . By Lemma 6.0, we conclude that each descendant clique of p_1 must contain $Z \cap R$. Thus we have $U(p_1) \supseteq Z \cap R \supseteq U(n_2) = U(n_1) \supseteq U(p_1)$. Hence all of these sets must be equal and hence we infer the following: $Z \cap R = U(n_1)$ and hence $U(x_{k_0}) \cap U(r_1) = U(n_1)$. Further, each subclique child of p_1 must precisely be the clique $U(n_1)$.

Since we have eliminated adjacent duplicates from all Q-nodes, there can be at most one such subclique in each tail of n_1 . Now if the subclique ($U(n_1)$) appears on both sides of n_1 , then there is a unique way of collapsing n_1 (see Figure 6(top)). Otherwise we collapse n_1 in such a way that $U(n_1)$ is present in the tail of x_{k_0} as shown in Figure 6(bottom). This is justified (i.e. it preserves compatibility between \mathcal{T}_1 and \mathcal{T}_2) because $U(n_1)$ can be inserted into the right tail of x_{k_0} in both \mathcal{T}_1 and \mathcal{T}_2 . In other words, if \mathcal{T}_1 and \mathcal{T}_2 are compatible, then there exists an intersection tree in which $U(n_1)$ is present in the right tail of x_{k_0} . The template reductions of this case, clearly run in $O(n)$ time.

Case 4.2.3: p_1 is a Q-node and has more than one essential child.

Let y be an essential child of p_1 , such that all the nodes between n_1 and y are subcliques. Without loss of generality, we assume that y appears to the right of n_1 . We collapse n_1 , depending on whether $MMD(y) \cap MMD(r_1)$ is empty or not, as shown in Figure 7. Thus the template reduction runs in $O(n)$ time.

If $MMD(y) \cap MMD(r_1)$ is non-empty, there exists a max-clique Y that is a descendant of both r_1 and y . Now if \mathcal{T}_1 and \mathcal{T}_2 are compatible, then in the leaf ordering of any intersection tree, the max-clique descendants of x_{k_0} appear in between the max-clique descendants of l_1 and Y . Thus we collapse the node n_1 , by deleting n_1 , and reassigning p_1 as the parent of all the children of n_1 . (Thus no essential node appears between x_{k_0} and y).

On the other hand if $MMD(y) \cap MMD(r_1)$ is empty, we observe the following: In the leaf ordering of any intersection tree \mathcal{T}_I no max-clique appears in between the max-clique descendants of x_{k_0} and the max-clique descendants of r_1 . Therefore, in this case we collapse n_1 , by reversing its children, deleting it,

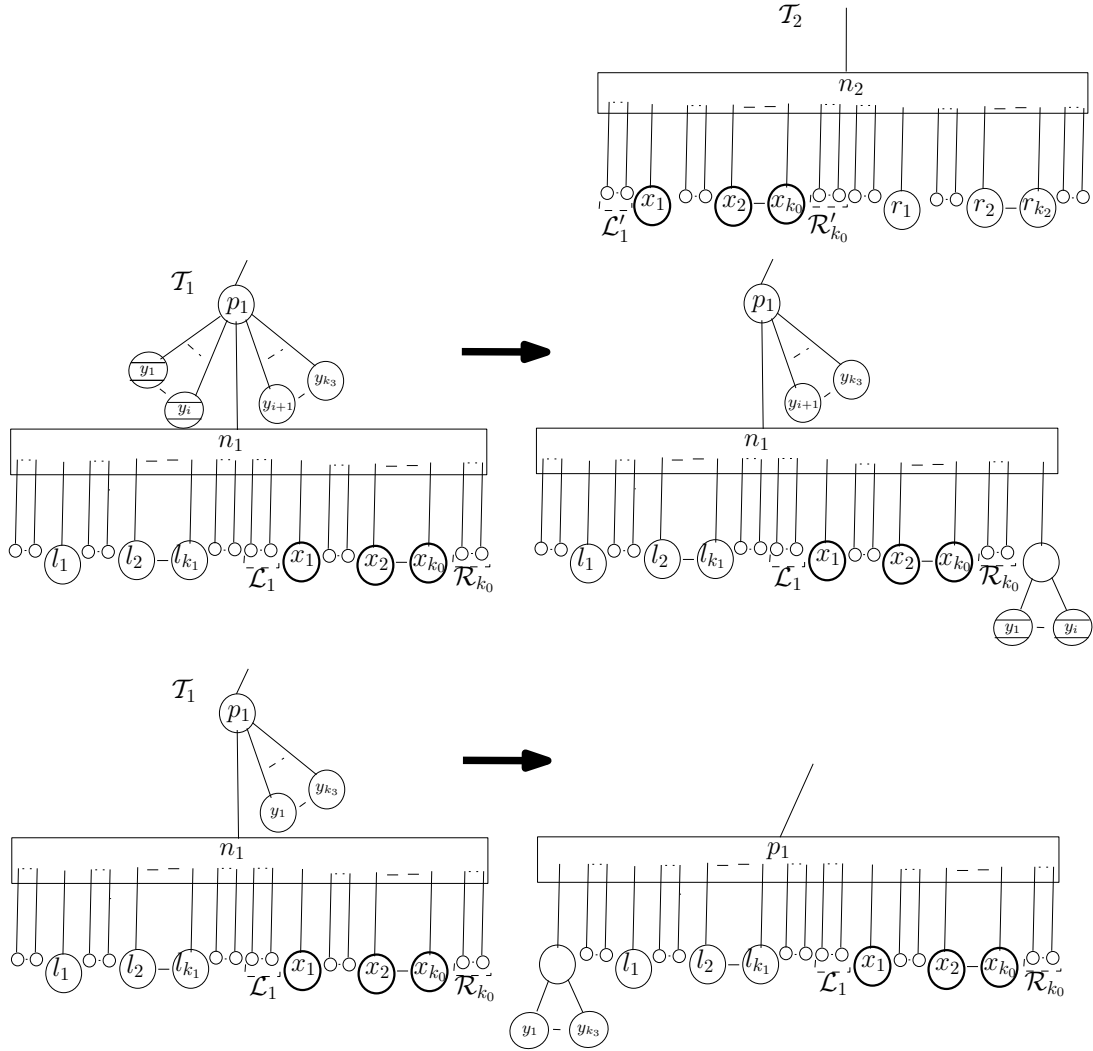


Figure 5: Reduction template of \mathcal{T}_1 for Case 4.2.1. A node y_a has horizontal stripes if $MMD(y_a) \cap MMD(r_b) \neq \emptyset$ for some r_b and no stripes otherwise.

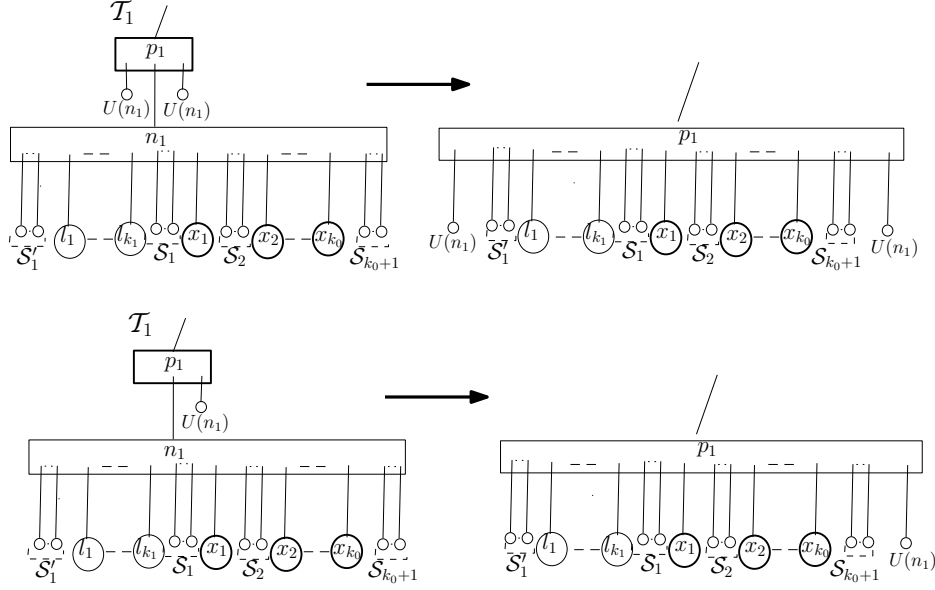


Figure 6: Reduction templates of \mathcal{T}_1 for Case 4.2.2.

and reassigning p_1 as the parent of all the children of n_1 . (Thus no essential node appears between l_1 and y).

Case 4.3: $U(n_1) \supseteq U(n_2)$ and $C_1 - X$ is empty

As before we have three cases depending on whether p_1 is a P-node or a Q-node and whether p_1 has more than one essential child. In each of these cases, when $|X| = 1$, we need to first solve the alignment problem (as a preprocessing step). Also when p_1 is a Q-node, unlike in Case 4.2, both ways of collapsing n_1 may lead to a valid intersection tree.

Alignment Problem

Recall that when $|X| = 1$ (and $C_1 - X = \emptyset$), the alignment may not be unique i.e. one of the following might happen in the intersection tree \mathcal{T}_I .

1. Left and right tails of x_1 (in \mathcal{T}_I) contain $\mathcal{L}_1 + \mathcal{L}'_1$ and $\mathcal{R}_1 + \mathcal{R}'_1$ respectively.
2. Left and right tails of x_1 (in \mathcal{T}_I) contain $\bar{\mathcal{R}}_1 + \mathcal{L}'_1$ and $\bar{\mathcal{L}}_1 + \mathcal{R}'_1$ respectively.

If one of the merges in (1) or (2) is invalid, then there is only a single way of aligning the tails, otherwise we show in the following Lemma that if \mathcal{T}_1 and \mathcal{T}_2 are compatible, then choosing either one of the two alignments will work.

Lemma 9. *Let $U(n_1) \supseteq U(n_2)$, $|X| = 1$ and $C_1 - X$ be empty. Let $\mathcal{L}_1, \mathcal{R}_1$ be the left and right tails of x_1 in \mathcal{T}_1 and $\mathcal{L}'_1, \mathcal{R}'_1$ be the left and right tails of x_1 in \mathcal{T}_2 . If both ways of alignment are mergable i.e. (a) $\mathcal{L}_1 + \mathcal{L}'_1, \mathcal{R}_1 + \mathcal{R}'_1$ are valid and (b) $\bar{\mathcal{R}}_1 + \mathcal{L}'_1, \bar{\mathcal{L}}_1 + \mathcal{R}'_1$ are valid, then there exists an intersection tree \mathcal{T}_I (of \mathcal{T}_1 and \mathcal{T}_2) with $\mathcal{L}_1 + \mathcal{L}'_1$ and $\mathcal{R}_1 + \mathcal{R}'_1$ contained in the left and right tails of x_1 (respectively) if and only if there exists an intersection tree \mathcal{T}'_I with $\bar{\mathcal{R}}_1 + \mathcal{L}'_1$ and $\bar{\mathcal{L}}_1 + \mathcal{R}'_1$ contained in the left and right tails of x_1 (respectively).*

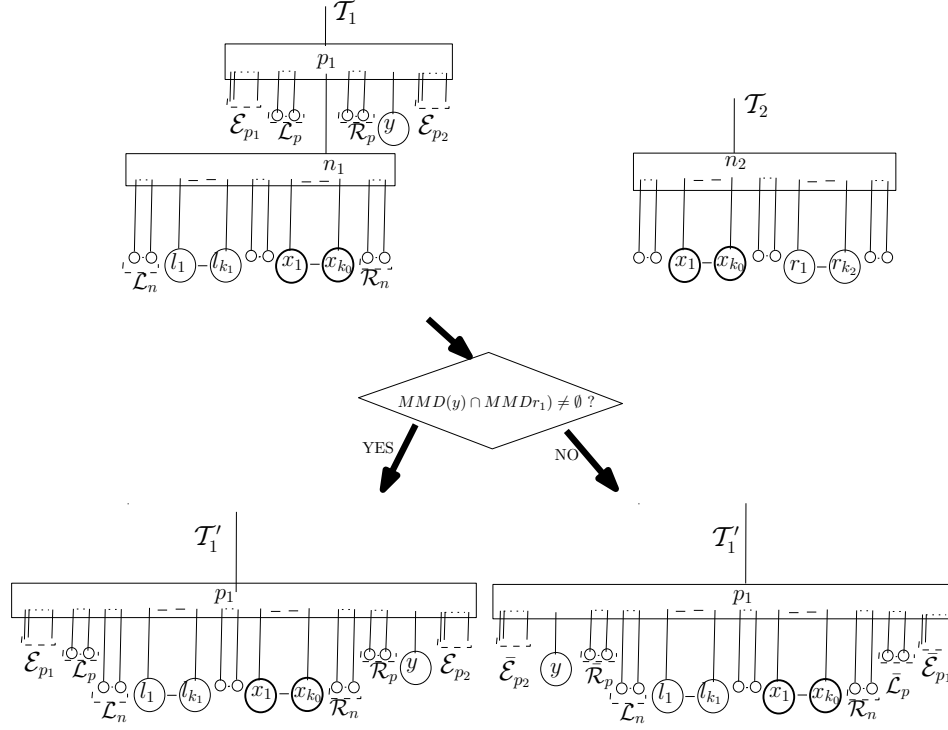


Figure 7: Reduction template of \mathcal{T}_1 for Case 4.2.3

Proof. Let \mathcal{L} , \mathcal{R} be the left and right tails of x_1 in an intersection tree \mathcal{T}_I . Each subclique S in \mathcal{L} or \mathcal{R} appears as a subclique in \mathcal{T}_1 or \mathcal{T}_2 . In particular we observe the following:

Property 1: If S is a subclique in \mathcal{L} or \mathcal{R} then in \mathcal{T}_1 or \mathcal{T}_2 , S is present in a tail of x_1 or in a tail of an ancestor of x_1 .

Note that since n contains at least two children, \mathcal{L} and \mathcal{R} both cannot be empty. If one of them, say \mathcal{L} is empty then the last clique in \mathcal{R} must be $U(n_1)$. If both \mathcal{L} and \mathcal{R} are non-empty then the intersection of the first subclique of \mathcal{L}_1 with the last subclique of \mathcal{R}_1 is $U(n_1)$. In either case we observe that, since $\mathcal{L}_1 + \mathcal{L}'_1$ and $\bar{\mathcal{R}}_1 + \mathcal{L}'_1$ are both valid (subclique orderings), each subclique in \mathcal{L}'_1 is either a superset of $U(n_1)$ or a subset of $U(n_1)$. Similarly, since $\mathcal{R}_1 + \mathcal{R}'_1$ and $\bar{\mathcal{L}}_1 + \mathcal{R}'_1$ are both valid (superclique orderings), each subclique in \mathcal{R}'_1 is either a superset of $U(n_1)$ or a subset of $U(n_1)$. Further for any ancestor n_a of n_2 , $U(n_a) \subseteq U(n_2) \subseteq U(n_1)$ and hence the tails of any such n_a would consist of subcliques that are subsets of $U(n_1)$. Note that this condition also holds for any ancestor of n_1 in \mathcal{T}_1 .

By above conditions and (1) we infer that for any subclique S in \mathcal{L} or \mathcal{R} , S is either a superset of $U(n_1)$ or a subset of $U(n_1)$. Furthermore, if S is a superset of $U(n_1)$ then it is present in one of $\mathcal{L}_1, \mathcal{R}_1, \mathcal{L}'_1$ or \mathcal{R}'_1 . This implies that if there exists an intersection tree \mathcal{T}_I in which \mathcal{L} contains $\mathcal{L}_1 + \mathcal{L}'_1$ and \mathcal{R} contains $\mathcal{R}_1 + \mathcal{R}'_1$, then replacing \mathcal{L} with $\mathcal{L} - \mathcal{L}_1 + \bar{\mathcal{R}}_1$ and \mathcal{R} with $\mathcal{R} - \mathcal{R}_1 + \bar{\mathcal{L}}_1$ also results in a valid intersection tree. \square

Note that the amortized cost of doing the mergability checks (a) and (b) of Lemma 9 (over all iterations of the algorithm) is $O(n \cdot |I|) = O(n^2)$. For the rest of the cases, we can assume that \mathcal{L}_1 is aligned with \mathcal{L}_2

and \mathcal{R}_1 is aligned with \mathcal{R}_2 . In other words if \mathcal{T}_1 and \mathcal{T}_2 are compatible, then there exists an intersection tree that contains $\mathcal{L}_1 + \mathcal{L}_2$ and $\mathcal{R}_1 + \mathcal{R}_2$ as the tails of x_1 .

Case 4.3.1: p_1 is a P-node.

If $C_2 - X = \emptyset$, then using the same argument as before (Lemma 6.0), we get $U(p_1) \subseteq U(n_2)$. Hence we replace \mathcal{L}_i and \mathcal{L}'_i with $\mathcal{L}_i + \mathcal{L}'_i$ in \mathcal{T}_1 and \mathcal{T}_2 changing $U(n_1)$ to $U(n_1) \cap U(n_2) \supseteq U(p_1)$, and we match n_1 with n_2 .

Now we look at the case when $C_2 - X$ is non-empty. Let $L = \{l_1, l_2, \dots, l_{k_1}\}$ be the set of essential nodes appearing to the left of X and $R = \{r_1, \dots, r_{k_2}\}$ be the set of essential nodes appearing to the right of X in \mathcal{T}_2 . Let $Y = \{y_1, \dots, y_{k_3}\}$ be all the remaining child nodes of p_1 other than n_1 . For all $i \in \{1, \dots, k_3\}$, if $MMD(y_i) \cap MMD(l_j) \neq \emptyset$ for some $j \in \{1, \dots, k_1\}$, then y_i and l_j both have a common max-clique descendant say Y , and further in any leaf order of a common intersection tree $\mathcal{L}_1 + \mathcal{L}'_1$ must appear between Y and the descendants of x_1 .

Thus we group all y_i such that $MMD(y_i) \cap MMD(l_j) \neq \emptyset$ into a new P-node and add it to the (immediate) left of \mathcal{L}_1 (see Figure 8). Similarly, we group all y_i such that $MMD(y_i) \cap MMD(r_j) \neq \emptyset$, for some $j \in \{1, \dots, k_2\}$ into a new P-node and add it to the (immediate) right of \mathcal{R}_{k_0} .

Note that if for some $y \in Y$, there exists l_i and r_j such that both $MMD(y) \cap MMD(l_i)$ and $MMD(y) \cap MMD(r_j) \neq \emptyset$, then we can conclude that \mathcal{T}_1 and \mathcal{T}_2 are incompatible.

Also if L and R are both non-empty and for all $y \in Y$, $MMD(y)$ doesn't intersect with any $MMD(l_i)$ for $i \in \{1, \dots, k_1\}$ and with any $MMD(r_j)$ for $j \in \{1, \dots, k_2\}$ then once again we conclude that \mathcal{T}_1 and \mathcal{T}_2 are incompatible.

On the other hand if one of L or R is empty, say L , and $MMD(y_i) \cap MMD(r_j)$ is empty for all $y_i \in Y$ and $r_j \in R$, then the above template would not reduce \mathcal{T}_1 . But then note that in any leaf-ordering of any intersection tree, $\mathcal{L}_1 + \mathcal{L}'_1$ should appear between the descendants of y_i and x_1 for all $y_i \in Y$ (because of the constraints imposed by \mathcal{T}_1 and \mathcal{T}_2). Hence in this case we group all the nodes of Y into a P-node and add it a child node of p_1 to the (immediate) left of \mathcal{L}_1 as shown in Figure 9.

Note that since the MM-Descendants of any two sibling nodes are disjoint, both of the above templates can be implemented in $O(n)$ time.

Case 4.3.2: p_1 is a Q-node and n_1 is its only essential child.

Let \mathcal{L}_p and \mathcal{R}_p be the left and right tails of p_1 . Note that in this case all the siblings of n_1 are subcliques that are present in its tails. We have three subcases depending on how $U(p_1)$ intersects $U(n_2)$.

Suppose $U(p_1)$ properly intersects $U(n_2)$. We have $U(p_1) - U(n_2) \neq \emptyset$ and $U(n_2) - U(p_1) \neq \emptyset$. We first claim that $C_2 - X$ is empty. Suppose not. Let Z be a max-clique descendant of a node in $C_2 - X$ and X_1 be a max-clique descendant of x_1 . By Lemma 6.2, in \mathcal{T}_1 , Z appears outside the subtree rooted at n_1 , and hence outside the subtree rooted at p_1 . Thus using Lemma 6.0, we conclude that each descendant of p_1 must contain all the vertices in $Z \cap X_1 \supseteq U(n_2)$. A contradiction. Hence $C_2 - X$ is empty.

Now by Lemma 6.6, there exists a subclique $S_1 \not\supseteq U(n_2)$ such that S_1 is the first clique of \mathcal{L}_p or the last clique of \mathcal{R}_p . Similarly there exists a subclique $S_2 \not\supseteq U(p_1)$ such that S_1 is the first clique of \mathcal{L}'_1 or the last clique of \mathcal{R}'_{k_0} . Without loss of generality let S_1 be the first clique of \mathcal{L}_p and S_2 be the last clique of \mathcal{R}'_{k_0} . Observe that $S_1 \supseteq U(p_1)$ and $S_2 \supseteq U(n_2)$. This implies that S_1 and S_2 cannot be in the same tail (of x_1 or x_{k_0}) in any intersection tree of \mathcal{T}_1 and \mathcal{T}_2 . Thus we reduce \mathcal{T}_1 by collapsing n_1 i.e. by deleting n_1 , changing the parent of child nodes of n_1 to p_1 and arranging the child nodes such that \mathcal{L}_p appears to the left of \mathcal{L}_1 and \mathcal{R}_p appears to the right of \mathcal{R}_{k_0} . Clearly, this reduction can be done in $O(n)$ time.

Now we have to deal with the case when either $U(n_2) \subseteq U(p_1)$ or $U(p_1) \subseteq U(n_2)$. Note that in any intersection tree \mathcal{T}_I (of \mathcal{T}_1 and \mathcal{T}_2), the cliques of $\mathcal{L}_1 + \mathcal{L}'_1$ appear in the left tail of x_1 and the cliques of

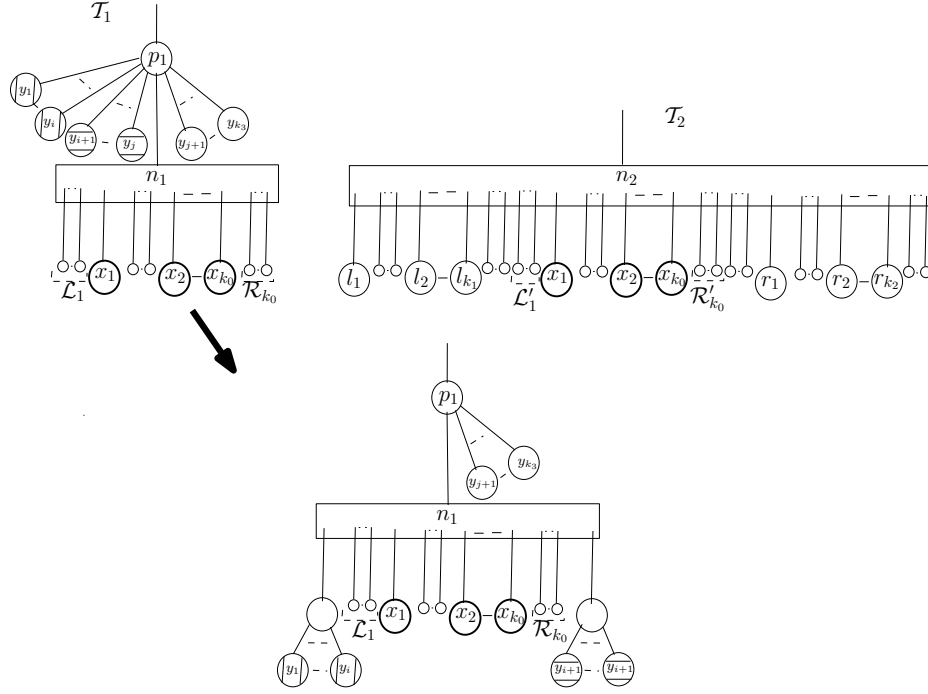


Figure 8: First reduction template of \mathcal{T}_1 for Case 4.3.1. A node y_a has vertical stripes if $MMD(y_a) \cap MMD(l_b) \neq \emptyset$ for some l_b , horizontal stripes if $MMD(y_a) \cap MMD(r_b) \neq \emptyset$ for some r_b and no stripes otherwise.

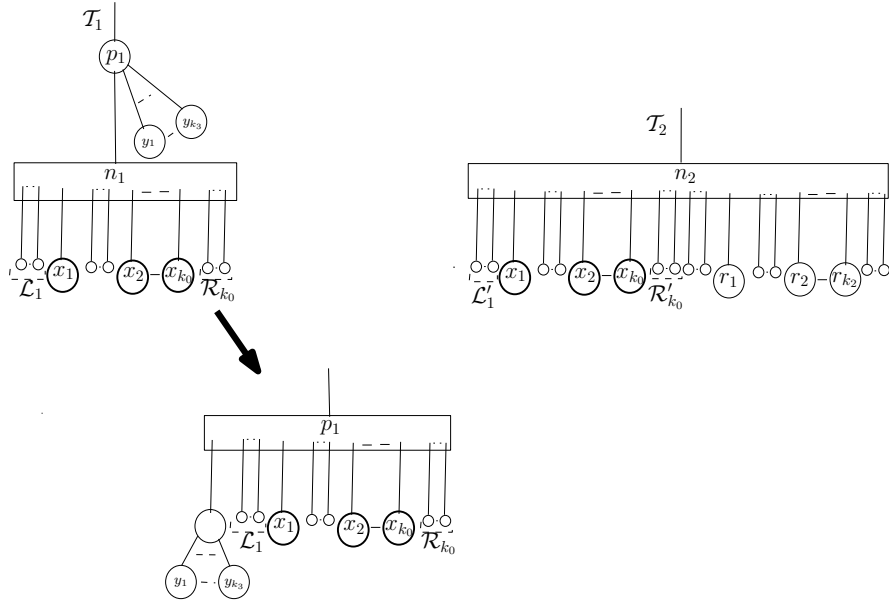


Figure 9: Second reduction template of \mathcal{T}_1 for Case 4.3.1. The stripes on the y nodes are defined as before

$\mathcal{R}_{k_0} + \mathcal{R}'_{k_0}$ appear in the right tail of x_{k_0} . Further either (a) the cliques of \mathcal{L}_p appear in the left tail of x_1 and the cliques of \mathcal{R}_p appear in the right tail of x_{k_0} or (b) the cliques of \mathcal{L}_p appear in the right tail of x_{k_0} and the cliques of \mathcal{R}_p appear in the left tail of x_1 . In the first case $\mathcal{L}_1 + \mathcal{L}'_1 + \mathcal{L}_p$ and $\mathcal{R}_{k_0} + \mathcal{R}'_{k_0} + \mathcal{R}_p$ are both valid and in the second case $\mathcal{L}_1 + \mathcal{L}'_1 + \bar{\mathcal{R}}_p$ and $\mathcal{R}_{k_0} + \mathcal{R}'_{k_0} + \bar{\mathcal{L}}_p$ are both valid. If neither of these is valid then we conclude that \mathcal{T}_1 and \mathcal{T}_2 are incompatible. If exactly one of the above merges is valid, then there is a unique way of collapsing n_1 . When both of the above merge pairs are valid, we use the reduction template shown in Figure 10. The justification (given below) depends on whether $U(n_2) \subseteq U(p_1)$ or $U(p_1) \subseteq U(n_2)$.

Let $U(n_2) \subseteq U(p_1)$. Note that by Lemma 6.6, the intersection of the universal nodes of the first and last child nodes of p_1 is $U(p_1)$. Hence if $\mathcal{L}_1 + \mathcal{L}'_1 + \mathcal{L}_p$, $\mathcal{R}_{k_0} + \mathcal{R}'_{k_0} + \mathcal{R}_p$, $\mathcal{L}_1 + \mathcal{L}'_1 + \bar{\mathcal{R}}_p$ and $\mathcal{R}_{k_0} + \mathcal{R}'_{k_0} + \bar{\mathcal{L}}_p$ are all valid then any subclique in \mathcal{L}'_1 or \mathcal{R}'_{k_0} is either a superset or a subset of $U(p_1)$. Thus in any intersection tree \mathcal{T}_I , any subclique S in the left tail of x_1 or the right tail of x_{k_0} is either a superset or a subset of $U(p_1)$. Further if $S \supseteq U(p_1)$, then S must appear in one of $\{\mathcal{L}'_1, \mathcal{R}'_{k_0}, \mathcal{L}_p, \mathcal{R}_p, \mathcal{L}_1, \mathcal{R}_{k_0}\}$. This implies that an intersection tree satisfying condition (a) exists if and only if an intersection tree satisfying condition (b) exists. This justifies the use of our template in Figure 10, for reducing \mathcal{T}_1 .

Similarly, if $U(p_1) \subseteq U(n_2)$, we infer that any clique in \mathcal{L}_p or \mathcal{R}_p is either a subset of $U(n_2)$ or a superset of $U(n_2)$. This in turn implies that in \mathcal{T}_I , any subclique S in the left tail of x_1 or the right tail of x_1 , is either a subset or a superset of $U(n_2)$. Further, if $S \supseteq U(n_2)$ then it must appear in $\{\mathcal{L}_1, \mathcal{R}_{k_0}, \mathcal{L}'_1, \mathcal{R}'_{k_0}, \mathcal{L}_p, \mathcal{R}_p\}$. This implies that an intersection tree satisfying condition (a) exists if and only if an intersection tree satisfying (b) exists. This justifies the use of our template in Figure 10, for reducing \mathcal{T}_1 .

We now show that the amortized cost of executing the reduction template in Figure 10, over all instances of the algorithm takes $O(n^2)$ time. Note that we use the same template for Case 4.3.3 when $C_2 - X$ is empty. It is enough to show that the amortized time of all the mergability checks: (whether $\mathcal{L}'_1 + \mathcal{L}_p$ and $\mathcal{R}'_{k_0} + \mathcal{R}_p$ are both valid) take $O(n^2)$ time.

Let $c(\mathcal{L}'_1)$ and $c(\mathcal{R}'_{k_0})$ be the (consecutive) subsequences of \mathcal{L}'_1 and \mathcal{R}'_{k_0} (respectively) such that each subclique in $c(\mathcal{L}'_1)$ and $c(\mathcal{R}'_{k_0})$ contains $U(p_1)$ but not $U(n_1)$. $c(\mathcal{L}'_1)$ and $c(\mathcal{R}'_{k_0})$ are said to be the core tails of n_2 .

Similarly let $c(\mathcal{L}_p)$ and $c(\mathcal{R}_p)$ be the (consecutive) subsequences of \mathcal{L}_p and \mathcal{R}_p (respectively) such that each subclique in $c(\mathcal{L}_p)$ and $c(\mathcal{R}_p)$ contains $U(n_2)$. $c(\mathcal{L}_p)$ and $c(\mathcal{R}_p)$ are said to be the core tails of p_1 .

Note that the core tails are only defined for p_1 and n_2 , for the current case and Case 4.3.3, when $C_2 - X$ is empty. We define the core tails of all other nodes to be empty. Observe that when n_1 is collapsed, the (new) core tails of any node in \mathcal{T}_1 (resp. \mathcal{T}_2) are disjoint from the core tails of p_1 (resp. n_2) before the collapse.

We observe that checking the validity of $\mathcal{L}'_1 + \mathcal{L}_p$ reduces to checking the validity of $c(\mathcal{L}'_1) + c(\mathcal{L}_p)$. Similarly, checking the validity of $\mathcal{R}'_{k_0} + \mathcal{R}_p$ reduces to checking the validity of $c(\mathcal{R}'_{k_0}) + c(\mathcal{R}_p)$.

Now computing the cores over all executions of this template, takes $O(n \cdot |I|) = O(n^2)$ amortized time. Also, computing the mergability of the cores, over all executions of the template takes $\sum_i (m_i + t_i) |I|$, where m_i, t_i are the number of subcliques in the cores of n_2 and p_1 (respectively), in the i th execution of the template. Since $\sum_i m_i = O(n)$ and $\sum_i t_i = O(n)$, the total running time of template 10 over all executions is $O(n^2)$.

Case 4.3.3: p_1 is a Q-node with more than one essential child.

Now let y be an essential child of p_1 , such that all the nodes between n_1 and y are subcliques. Without loss of generality, we assume that y appears to the right of n_1 .

We first consider the subcase when $C_2 - X$ is empty. In this case observe that all the max-clique descendants of y appear outside the subtree rooted at n_1 in \mathcal{T}_1 . Applying Lemma 6.0 on a max-clique

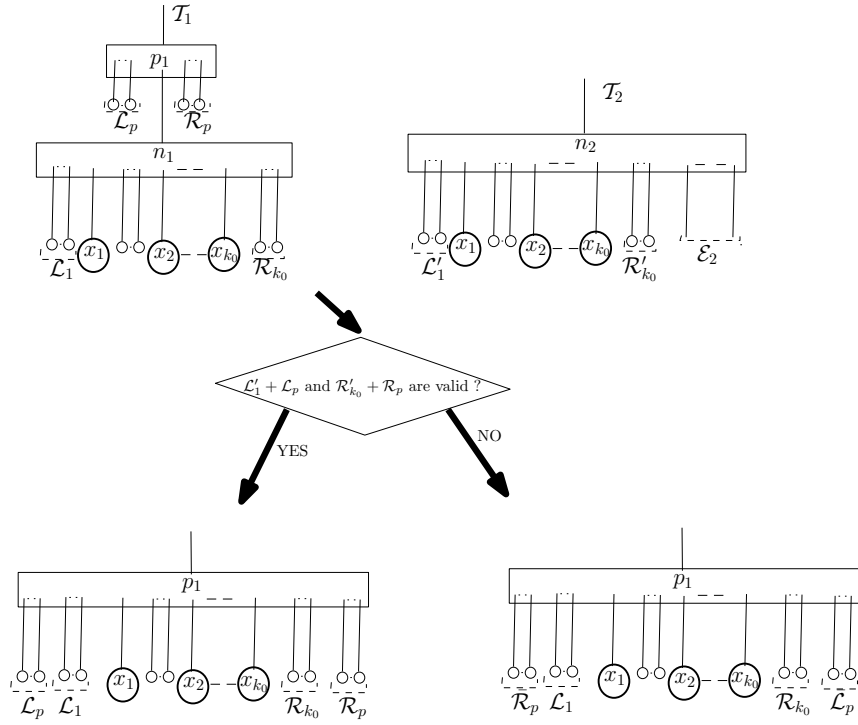


Figure 10: Reduction template of \mathcal{T}_1 for Case 4.3.2. Note that \mathcal{E}_2 denotes the (possibly empty) sequence of children of n_2 that appear after \mathcal{R}'_{k_0} .

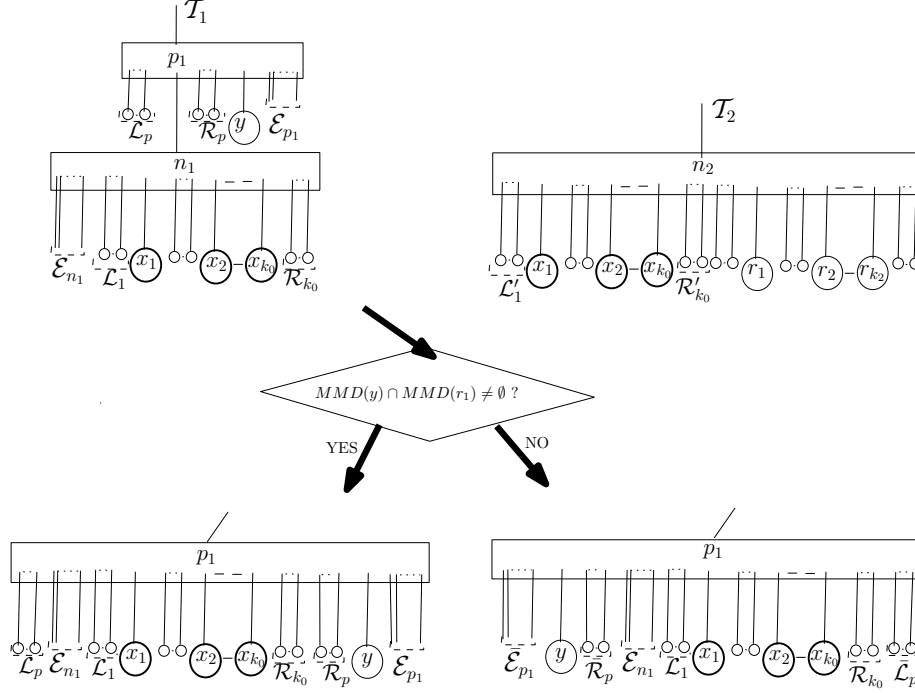


Figure 11: Reduction template of \mathcal{T}_1 for Case 4.3.3.

descendant of y and a max-clique descendant of x_1 , we infer that each descendant clique of n_2 must contain all the vertices in $U(p_1)$. In other words we get $U(p_1) \subseteq U(n_2)$. Now the template (and the argument) in this case is analogous to case 4.3.2, when $U(p_1) \subseteq U(n_2)$ (Figure 10).

Now suppose $C_2 - X$ is non empty. Let r_1 be the first essential child to the right of x_{k_0} in \mathcal{T}_2 . We use the templates in Figure 11, depending on whether $MMD(y) \cap MMD(r_1)$ is empty or not. If $MMD(y) \cap MMD(r_1)$ is non-empty, then by Lemma 6.2, there exists a max-clique Y that is a descendant of both r_1 and y . Now if \mathcal{T}_1 and \mathcal{T}_2 are compatible, then in any leaf ordering of an intersection tree \mathcal{T}_I , the subcliques of $\mathcal{R}_{k_0} + \mathcal{R}'_{k_0}$ appear in between the descendants of x_k and Y (because of \mathcal{T}_2). This justifies the reduction template of \mathcal{T}_1 in Figure 11.

On the other hand, if $MMD(r_1) \cap MMD(y) = \emptyset$, then by the constraints of \mathcal{T}_2 , in any leaf ordering of \mathcal{T}_I , the subcliques of $\mathcal{R}_{k_0} + \mathcal{R}'_{k_0}$ appear between the descendants of x_{k_0} and l_1 and further no max-clique appears between them. This justifies the reduction template of \mathcal{T}_1 in Figure 11. Moreover the template reduction takes $O(n)$ time.

Run time of the Algorithm

In this Section, we show that the run time of our algorithm is $O(n^2 \log n)$, where n is the total number of vertices in $G_1 \cup G_2$.

Observe that reducing \mathcal{T}_1 [resp. \mathcal{T}_2] decreases the number of leaf orderings of \mathcal{T}_1 [resp. \mathcal{T}_2] by at least half. Moreover the total number of nodes in \mathcal{T}_1 and \mathcal{T}_2 is at most n . Thus the number of leaf orderings of \mathcal{T}_1 and \mathcal{T}_2 is at most $n!$ and hence the algorithm requires at most $n \log n$ reductions.

We begin by showing that selecting the nodes n_1 and n_2 takes $O(n)$ time in any iteration. We first note that computing the number of MM-Descendants for all the nodes takes $O(n)$ time (they can be computed in a bottom-up fashion). With each node x , we store $U(x)$ and the cardinality of $MMD(x)$.

Recall that as we go down the tree the universal sets increase and the MM-Descendants sets decrease. Thus n_1 must have maximal depth among all unmatched essential nodes. Hence we can select n_1 in $O(n)$ time by looking at the unmatched essential nodes of maximal depth in \mathcal{T}_1 and \mathcal{T}_2 , and selecting a node with the greatest universal set size and the least number of MM-Descendants in that order. Note that by property 1 of Lemma 8, the MM-descendants of n_1 are same as the essential child nodes of n_1 . Now we can select n_2 from the other tree \mathcal{T}_2 in $O(n)$ time as follows: Let S be the set of [matched] essential children of n_1 and S' be the corresponding set of matched nodes in \mathcal{T}_2 . Let $p(S')$ be the set of parent nodes of nodes in S' . We select n_2 to be a node of maximum depth among $p(S')$.

Also recall that at the high-level our algorithm has 4 cases depending on whether n_1, n_2 are P-nodes or Q-nodes. We showed that each step in cases 1,2 or 3 takes $O(n)$ time. For case 4, we showed that each of reduction steps, excluding the mergability checks in Cases 4.3.2 and 4.3.3 take $O(n)$ time. We also showed that the mergability checks of Cases 4.3.2 and 4.3.3 take $O(n^2)$ amortized time over all steps of the algorithm. Further, in the beginning of Case 4, we showed that the matching steps, which involve inserting the subcliques of one tree into the other take at most $O(n^2)$ amortized time. Thus the total time taken by our algorithm is $O(n^2 \log n + n^2 + n^2) = O(n^2 \log n)$. At each node y of \mathcal{T}_1 (resp. \mathcal{T}_2) we explicitly store the set $U(y)$ and the cardinality of $MMD(y)$. Since the number of internal nodes is less than the number of leaf nodes, this additional storage still takes $O(n + m)$. Thus the space complexity of our algorithm is $O(n + m)$.

5 Open Problem

Simultaneous graphs can be generalized in a natural way to more than two graphs: when $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \dots, G_k = (V_k, E_k)$ are k graphs in class \mathcal{C} , sharing a vertex set I and its induced edges i.e. $V_i \cap V_j = I$ for all $i, j \in \{1, \dots, k\}$. In this version of the problem the set of optional edges induces a complete k -partite graph and hence this also generalizes probe graphs. This generalized version can be solved in polynomial time for comparability and permutation graphs [9]. We conjecture that it can be solved in polynomial time for interval graphs.

References

- [1] Berry, A., Golumbic, M.C., Lipshteyn, M. Recognizing chordal probe graphs and cycle-bicolorable graphs. *SIAM J. Discret. Math.*, 21(3):573–591, 2007.
- [2] Booth, K.S and Lueker, G.S. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.
- [3] 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. Theory Appl.*, 36(2), 117-130 (2007).
- [4] Estrella-Balderrama, A., Gassner, E., Junger, M., Percan, M., Schaefer, M., Schulz, M., Simultaneous geometric graph embeddings. *GD 2007*. LNCS, vol. 4875, pp.280-290.
- [5] Chandler, D.B., Chang, M., Kloks, T., Liu, J., and Peng, S. Partitioned probe comparability graphs. *Theor. Comput. Sci.*, 396(1-3):212–222, 2008.
- [6] Golumbic, M.C. *Algorithmic Graph Theory And Perfect Graphs*. Academic Press, New York, 1980.

- [7] Golumbic, M.C., Kaplan, H., and Shamir, R. Graph sandwich problems. *J. Algorithms*, 19(3):449–473, 1995.
- [8] Golumbic M.C, and Lipshteyn, M. Chordal probe graphs. *Discrete Appl. Math.*, 143(1-3):221–237, 2004.
- [9] Jampani, K.R., Lubiw, A. The simultaneous representation problem for chordal, comparability and permutation graphs. In *WADS '09*, pages 387–398, 2009.
- [10] Johnson, J.L., and Spinrad, J.P. A polynomial time recognition algorithm for probe interval graphs. In *SODA '01*, pages 477–486, Philadelphia, PA, USA, 2001.
- [11] McConnell, R.M., and Nussbaum, Y. Linear-time recognition of probe interval graphs. In *ESA 09*, pages 349–360.
- [12] McConnell, R.M., and Spinrad, J.P. Construction of probe interval models. In *SODA '02*, pages 866–875, Philadelphia, PA, USA, 2002.
- [13] McMorris, F. R., Wang, C., Zhang, P. On probe interval graphs. *Discrete Appl. Math.*, 88(1-3):315–324, 1998.
- [14] Tanenbaum, Paul J. Simultaneous intersection representation of pairs of graphs. *J. Graph Theory.*, 32(2):171–190, 1999.