Canonical Data Structure for Probe Interval Graphs

Ryuhei Uehara*

2004/2/7

Abstract

The class of probe interval graphs is introduced to deal with the physical mapping and sequencing of DNA as a generalization of interval graphs. The polynomial time recognition algorithms for the graph class are known. However, the complexity of the graph isomorphism problem for the class is still unknown. In this paper, extended $\mathcal{MPQ}$-trees are proposed to represent the probe interval graphs. An extended $\mathcal{MPQ}$-tree is canonical and represents all possible permutations of the intervals. The extended $\mathcal{MPQ}$-tree can be constructed from a given probe interval graph in $O(n^2+m)$ time. Thus we can solve the graph isomorphism problem for the probe interval graphs in $O(n^2+m)$ time. Using the tree, we can determine that any two nonprobes are independent, overlapping, or their relation cannot be determined without an experiment. Therefore, we can heuristically find the best nonprobe that would be probed in the next experiment. Also, we can enumerate all possible affirmative interval graphs for given probe interval graph.
Keywords: Bioinformatics, graph isomorphism, probe interval graph.

1 Introduction

The class of interval graphs was introduced in the 1950's by Hajös and Benzer independently. Since then a number of interesting applications for interval graphs have been found including to model the topological structure of the DNA molecule, scheduling, and others (see [8, 15, 5] for further details). The interval graph model requires all overlap information. However, in many cases, only partial overlap data exist. The class of probe interval graphs is introduced by Zhang in the assembly of contigs in physical mapping of DNA, which is a problem arising in the sequencing of DNA (see [18, 20, 19, 15] for background). A probe interval graph is obtained from an interval graph by designating a subset P of vertices as *probes*, and removing the edges between pairs of vertices in the remaining set N of *nonprobes*. That is, on the model, only partial overlap information (between a probe and the others) is given. A few efficient algorithms for the class are known; the recognition algorithms [11, 14, 10], and an algorithm for finding a tree 7-spanner (see [4] for details). The recognition algorithm in [11] also gives a data structure that represents all possible permutations of the intervals of a probe interval graph.

A data structure called $\mathcal{PQ}$-trees was developed by Booth and Lueker to represent all possible permutations of the intervals of an interval graph [3]. Korte and Möhring simplified their algorithm by introducing $\mathcal{MPQ}$-trees [12]. An $\mathcal{MPQ}$-tree is canonical; that is, given two interval graphs are isomorphic if and only if their corresponding $\mathcal{MPQ}$-trees are isomorphic. However, there are no canonical $\mathcal{MPQ}$-trees for probe interval graphs. In general, given probe interval graph, there are several affirmative interval graphs those are not isomorphic, and their interval representations are consistent to the probe interval graph.

In this paper, we extend $\mathcal{MPQ}$-trees to represent probe interval graphs. The extended $\mathcal{MPQ}$-tree is canonical for any probe interval graph, and the tree can be constructed in $O(n^2+nm)$ time. Thus the graph isomorphism problem for probe interval graphs can be solved in $O(n^2+nm)$ time. From the theoretical point of view, the complexity of the graph isomorphism of probe interval graphs was not known (see [17] for related results and references). Thus the result improves the upper bound of the graph classes such that the graph isomorphism problem can be solved in polynomial time.

From the practical point of view, the extended $\mathcal{MPQ}$-tree is very informative, which is beneficial in the Computational Biology community. The extended $\mathcal{MPQ}$-tree gives us the information between nonprobes; the

*Natural Science Faculty, Komazawa University, Tokyo 154-8525, Japan. uehara@komazawa-u.ac.jp.

relation of two nonprobes is either (1) independent (they cannot overlap with each other), (2) overlapping, or (3) not determined without experiments. Hence, to clarify the structure of the DNA sequence, we only have to experiment on the nonprobes in the case (3). Moreover, given extended $\mathcal{MPQ}$-tree, we can find the nonprobe v that has most nonprobes u such that v and u are in the case (3) in linear time. Therefore, we can heuristically find the "best" nonprobe to fix the structure of the DNA sequence.

The extended $\mathcal{MPQ}$-tree also represents all possible permutations of the intervals of a probe interval graph as in [11].

2 Preliminaries

The *neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N_G(v) = \{u \in V | \{u, v\} \in E\}$, and the *degree* of a vertex v is $|N_G(v)|$ and denoted by $deg_G(v)$. For the vertex set U of V, we denote by $N_G(U)$ the set $\{v \in V | v \in N(u) \text{ for some } u \in U\}$. If no confusion can arise we will omit the index G. Given graph $G = (V, E)$, its *cograph* is defined by $\bar{E} = \{\{u, v\} | u, v \in V \text{ and } \{u, v\} \notin E\}$, and denoted by $\bar{G} = (V, \bar{E})$. A vertex set I is *independent set* if $G[I]$ contains no edges, and then the graph $\bar{G}[I]$ is said to be a *clique*.

For a given graph $G = (V, E)$, a sequence of the vertices $v_0, v_1, \cdots, v_l$ is a *path*, denoted by $(v_0, v_1, \cdots, v_l)$, if $\{v_j, v_{j+1}\} \in E$ for each $0 \le j \le l - 1$. The *length* of a path is the number of edges on the path. For two vertices u and v, the *distance* of the vertices is the minimum length of the paths joining u and v. A *cycle* is a path beginning and ending with the same vertex. An edge which joins two vertices of a cycle but is not itself an edge of the cycle is a *chord* of that cycle. A graph is *chordal* if each cycle of length at least 4 has a chord. Given graph $G = (V, E)$, a vertex $v \in V$ is *simplicial* in G if $G[N(v)]$ is a clique in G. The following lemma is a folklore:

Lemma 1 Given chordal graph, all simplicial vertices can be found in linear time.

Proof. It is well known that a graph $G = (V, E)$ is chordal if and only if it is an intersection graph of subtrees of a tree $T(G)$. It can be assumed that the vertices of $T(G)$ are the maximal cliques of G, and the subtrees T_v for $v \in V$ are defined by the occurrences of v in the maximal cliques of G. Such a tree $T(G)$ is called a *clique tree* of G, and can be found in linear time [9, 2, 7] (see [5, p. 7] for further details). In the clique tree $T(G)$, each vertex v is simplicial if and only if v appears exactly once in $T(G)$. Thus we can find all simplicial vertex in linear time. ∎

The ordering $(v_1, \cdots, v_n)$ of the vertices of V is a *perfect elimination ordering* of G if the vertex v_i is simplicial in $G[\{v_i, v_{i+1}, \cdots, v_n\}]$ for all $i = 1, \cdots, n$. Then a graph is chordal if and only if it has a perfect elimination ordering (see, e.g., [5, Section 1.2] for further details).

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if and only if there is a one-to-one mapping $\phi : V \to V'$ which satisfies $\{u, v\} \in E$ if and only if $\{\phi(u), \phi(v)\} \in E'$ for every pair of vertices u and v. We denote by $G \sim G'$ if G is isomorphic to G'. The mapping ϕ is called an *isomorphism* from G to G'. Given graphs G and G', *graph isomorphism problem* is the problem to determine if $G \sim G'$.

2.1 Interval graph representation

A graph (V, E) with $V = \{v_1, v_2, \cdots, v_n\}$ is an *interval graph* if there is a set of intervals $\mathcal{I} = \{I_{v_1}, I_{v_2}, \cdots, I_{v_n}\}$ such that $\{v_i, v_j\} \in E$ if and only if $I_{v_i} \cap I_{v_j} \neq \emptyset$ for each i and j with $1 \le i, j \le n$. We call the set $\mathcal{I}$ of intervals *interval representation* of the graph. For each interval I, we denote by $R(I)$ and $L(I)$ the right and left endpoints of the interval, respectively (hence we have $L(I) \le R(I)$ and $I = [L(I), R(I)]$).

A graph $G = (V, E)$ is a *probe interval graph* if V can be partitioned into subsets P and N (corresponding to the *probes* and *nonprobes*) and each $v \in V$ can be assigned to an interval I_v such that $\{u, v\} \in E$ if and only if both $I_u \cap I_v \neq \emptyset$ and at least one of u and v is in P. In this paper, we assume that P and N are given, and then we denote by $G = (P, N, E)$. By definition, N is an independent set, $G[P]$ is an interval graph, and $G[P \cup \{v\}]$ is also an interval graph for any $v \in N$. Let $G = (P, N, E)$ be a probe interval graph. Let E^+ be a set of edges $\{t_1, t_2\}$ with $t_1, t_2 \in N$ such that there are two probes v_1 and v_2 in P such that $\{v_1, t_1\} \in E$, $\{v_1, t_2\} \in E$, $\{v_2, t_1\} \in E$, $\{v_2, t_2\} \in E$, and $\{v_1, v_2\} \notin E$. Intuitively, nonprobes t_1 and t_2 are joined by an edge in E^+ if (1) there are two independent probes v_1 and v_2, and (2) both of v_1 and v_2 overlap t_1 and t_2. In the case, we can know that intervals t_1 and t_2 have to overlap without experiment. Each edge in E^+ is called an *enhanced edge*, and the graph $G^+ := (P, N, E \cup E^+)$ is said to be an *enhanced probe interval graph*. It is known that a probe interval graph is weakly chordal [16], and an enhanced probe interval graph is chordal [18, 20]. For further details and references can be found in [5, 15].

For given probe interval graph G, an interval graph G' is said to be *affirmative* if and only if G' gives one possible interval representations for G. More precisely, given probe interval graph $G = (P, N, E)$, an interval graph $G' = (V, E')$ is affirmative if and only if there is a partition of V into P and N such that $E = \{\{u, v\} | \{u, v\} \in E'$ and at least one of u and v is in $P\}$. In general, for a probe interval graph G, there are several non-isomorphic affirmative interval graphs. For given probe interval graph $G = (P, N, E)$, the affirmative interval graph G' is also said to be *affirmative* to the corresponding enhanced probe interval graph $G^+ = (P, N, E \cup E^+)$.

2.2 $\mathcal{PQ}$-trees and $\mathcal{MPQ}$-trees

$\mathcal{PQ}$-trees were introduced by Booth and Lueker [3], and which can be used to recognize interval graphs as follows. A *$\mathcal{PQ}$-tree* is a rooted tree T with two types of internal nodes: $\mathcal{P}$ and $\mathcal{Q}$, which will be represented by circles and rectangles, respectively. The leaves of T are labeled 1-1 with the maximal cliques of the interval graph G. The *frontier* of a $\mathcal{PQ}$-tree T is the permutation of the maximal cliques obtained by the ordering of the leaves of T from left to right. $\mathcal{PQ}$-tree T and T' are *equivalent*, if one can be obtained from the other by applying the following rules a finite number of times; (1) arbitrarily permute the successor nodes of a $\mathcal{P}$-node, or (2) reverse the order of the successor nodes of a $\mathcal{Q}$-node. In [3], Booth and Lueker showed that a graph G is an interval graph if and only if there is a $\mathcal{PQ}$-tree T whose frontier represents a consecutive arrangement of the maximal cliques of G. They also developed an $O(|V| + |E|)$ algorithm that either constructs a $\mathcal{PQ}$-tree for G, or states that G is not an interval graph. If G is an interval graph, then all consecutive arrangements of the maximal cliques of G are obtained by taking equivalent $\mathcal{PQ}$-trees.

Lueker and Booth [13], and Colbourn and Booth [6] developed labeled $\mathcal{PQ}$-trees in which each node contains information of vertices as labels. Their labeled $\mathcal{PQ}$-trees are *canonical*; given interval graphs G_1 and G_2 are isomorphic if and only if corresponding labeled $\mathcal{PQ}$-trees T_1 and T_2 are isomorphic. Since we can determine if two labeled $\mathcal{PQ}$-trees T_1 and T_2 are isomorphic, the isomorphism of interval graphs can be determined in linear time.

$\mathcal{MPQ}$-trees, which stands for *modified $\mathcal{PQ}$-trees*, are developed by Korte and Möhring to simplify the construction of $\mathcal{PQ}$-trees [12].The $\mathcal{MPQ}$-tree T^* assigns sets of vertices (possibly empty) to the nodes of a $\mathcal{PQ}$-tree T representing an interval graph $G = (V, E)$. A $\mathcal{P}$-node is assigned only one set, while a $\mathcal{Q}$-node has a set for each of its sons (ordered from left to right according to the ordering of the sons).

For a $\mathcal{P}$-node $\hat{P}$, this set consists of those vertices of G contained in all maximal cliques represented by the subtree or $\hat{P}$ in T, but in no other cliques[†]. For a $\mathcal{Q}$-node $\hat{Q}$, the definition is more involved. Let $Q_1, \cdots, Q_m$ ($m \geq 3$) be the set of the sons (in consecutive order) of $\hat{Q}$, and let T_i be the subtree of T with root Q_i. We then assign a set S_i, called *section*, to $\hat{Q}$ for each Q_i. Section S_i contains all vertices that are contained in all maximal cliques of T_i and some other T_j, but not in any clique belonging to some other subtree of T that is not below $\hat{Q}$.

In [12], Korte and Möhring showed two algorithms that construct an $\mathcal{MPQ}$-tree for given interval graph. The first one constructs an $\mathcal{MPQ}$-tree for given interval graph using its $\mathcal{PQ}$-tree. The $\mathcal{MPQ}$-tree directly corresponds to the labeled $\mathcal{PQ}$-tree; the sets of vertices assigned in the $\mathcal{MPQ}$-tree directly correspond to the "characteristic nodes" in [6]. Since the labeled $\mathcal{PQ}$-tree is canonical, so is the constructed $\mathcal{MPQ}$-tree. The second algorithm constructs an $\mathcal{MPQ}$-tree from given interval graph directly without constructing $\mathcal{PQ}$-trees in [3]. Although it does not shown explicitly, the $\mathcal{MPQ}$-tree constructed by the second algorithm is the same as the $\mathcal{MPQ}$-tree by the first algorithm. Thus the graph isomorphism problem can be solved in linear time using the $\mathcal{MPQ}$-trees, which can be obtained without constructing $\mathcal{PQ}$-trees in [3].

The property of $\mathcal{MPQ}$-trees is summarized as follows [12, Theorem 2.1]:

Theorem 2 Let T^* be the canonical $\mathcal{MPQ}$-tree for given interval graph $G = (V, E)$. Then

(a) T^* can be obtained in $O(|V| + |E|)$ time and $O(|V|)$ space.

(b) Each maximal clique of G corresponds to a path in T^* from the root to a leaf, where each vertex $v \in V$ is as close as possible to the root.

[†]We will use $\hat{P}, \hat{Q}$, and $\hat{N}$ for describing a $\mathcal{P}$-node, $\mathcal{Q}$-node, any node, respectively to distinguish probe set P and nonprobe set N.

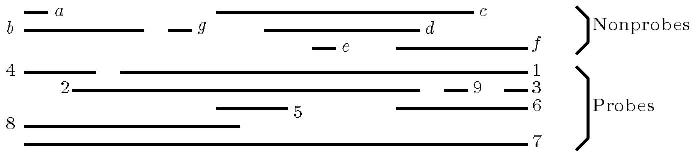

Figure 1: Given probe interval graph G

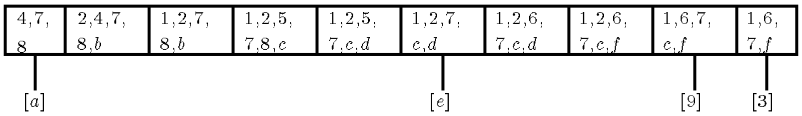

Figure 2: The $\mathcal{MPQ}$-tree of $G - g$

(c) In T^*, each vertex v appears in either one leaf, one $\mathcal{P}$-node, or consecutive sections S_i, S_{i+1}, $\cdots$, S_{i+j} (with $j > 0$) in a $\mathcal{Q}$-node.

(d) The root of T^* contains all vertices belonging to all maximal cliques, while the leaves contain the simplicial vertices.

Proof. The claims (a) and (b) are stated in [12, Theorem 2.1]. The claim (c) is immediately obtained by the fact that the maximal cliques containing a fixed vertex occur consecutively in T; see [12] for further details. The claim (d) is also stated in [12, p. 71]. ∎

We note that there are no vertices that appear in only one section in a $\mathcal{Q}$-node (since such a vertex appears in the subtree of the section). Thus each vertex in a $\mathcal{Q}$-node appears at least two consecutive sections.

Lemma 3 Let $\hat{Q}$ be a $\mathcal{Q}$-node in the canonical $\mathcal{MPQ}$-tree. Let $S_1, \cdots, S_k$ (in this order) be the sections of $\hat{Q}$, and let U_i denote the set of vertices occurring below S_i with $1 \leq i \leq k$. Then we have the following;

(a) $S_{i-1} \cap S_i \neq \emptyset$ for $2 \leq i \leq k$,

(b) $S_1 \subseteq S_2$ and $S_k \subseteq S_{k-1}$,

(c) $U_1 \neq \emptyset$ and $U_k \neq \emptyset$,

(d) $(S_i \cap S_{i+1}) \setminus S_1 \neq \emptyset$ and $(S_{i-1} \cap S_i) \setminus S_k \neq \emptyset$ for $2 \leq i \leq k-1$,

(e) $S_{i-1} \neq S_i$ with $2 \leq i \leq k-1$, and

(f) $(S_{i-1} \cup U_{i-1}) \setminus S_i \neq \emptyset$ and $(S_i \cup U_i) \setminus S_{i-1} \neq \emptyset$ for $2 \leq i \leq k$.

Proof. The results in [12] lead us from (a) to (e) immediately. Thus we show (f). If $(S_{i-1} \cup U_{i-1}) \setminus S_i = \emptyset$, we have $U_{i-1} = \emptyset$ and $S_{i-1} \subset S_i$. In the case, S_{i-1} is redundant section; we can obtain more compact $\mathcal{MPQ}$-tree by removing S_{i-1}. This fact contradicts that the $\mathcal{MPQ}$-tree is canonical. Thus (f) is settled. ∎

Given enhanced probe interval graph $G^+ = (P, N, E \cup E^+)$, let u and v be any two nonprobes with $\{u, v\} \notin E^+$. Then, we say that u *intersects* v if $I_u \cap I_v \neq \emptyset$ for all affirmative interval graphs of G^+. The nonprobes u and v are *independent* if $I_u \cap I_v = \emptyset$ for all affirmative interval graphs of G^+. Otherwise, we say that the nonprobe u *potentially intersects* v. Intuitively, if u potentially intersects v, we cannot determine their relation without experiments.

We define *depth* of each node in an $\mathcal{MPQ}$-tree as follows: the root has depth 0, and the other node has depth $d + 1$, where d is the depth of its parent. We also define *depth* of a vertex in G by the depth of the node in the $\mathcal{MPQ}$-tree that contains v. By Theorem 2(c), the depth of a vertex v in the $\mathcal{MPQ}$-tree is uniquely determined, and we denote by $dep(v)$.

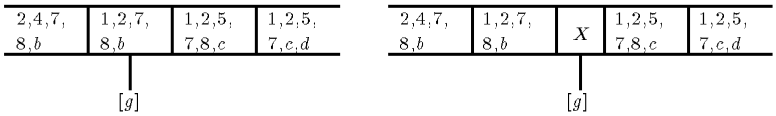

Figure 3: Four $\mathcal{MPQ}$-trees of G

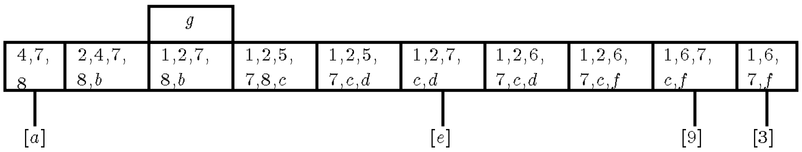

Figure 4: The extended $\mathcal{MPQ}$-tree of G

2.3 Extended $\mathcal{MPQ}$-trees

If given graph is an interval graph, the corresponding $\mathcal{MPQ}$-tree is uniquely determined up to isomorphism. However, for a probe interval graph, this is not in the case. For example, consider a probe interval graph $G = (P, N, E)$ with $P = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $N = \{a, b, c, d, e, f, g\}$ given in Figure 1. If the graph does not contain the nonprobe g, we have the canonical $\mathcal{MPQ}$-tree in Figure 2. However, the graph is a probe interval graph and we do not know if g intersects b and/or c since they are nonprobes. According to the relations between g and b and/or c, we have four possible $\mathcal{MPQ}$-trees that are affirmative to G shown in Figure 3, where X is either $\{1, 2, 7, 8\}$, $\{1, 2, 7, 8, c\}$, or $\{1, 2, 7, 8, b, c\}$. We call such a vertex g *floating leaf* (later, it will be shown that such a vertex has to be a leaf in an $\mathcal{MPQ}$-tree). For a floating leaf, there is a corresponding $\mathcal{Q}$-node (which also will be shown later). Thus we extend the notion of a $\mathcal{Q}$-node to contain the information of the floating leaf. A floating leaf appears consecutive sections of a $\mathcal{Q}$-node $\hat{Q}$ as the ordinary vertices in $\hat{Q}$. To distinguish them, we draw them over the corresponding sections; see Figure 4. Further details will be discussed in Section 3.

3 Construction of Extended $\mathcal{MPQ}$-tree of Probe Interval Graph

Let $G = (P, N, E)$ be a given probe interval graph, and $G^+ = (P, N, E \cup E^+)$ be the corresponding enhanced probe interval graph, where E^+ is the set of enhanced edges. In our algorithm, simplicial nonprobes play an important role; we partition the set N of nonprobes to two sets N^* and N_S defined as follows;

$$\begin{aligned} N_S &= \{u | u \text{ is simplicial in } G^+\}, \\ N^* &= N \setminus N_S. \end{aligned}$$

For example, for the graph $G = (P, N, E)$ in Figure 1, $E^+ = \{\{c, d\}, \{c, f\}\}$, $N_S = \{a, e, g\}$, and $N^* = \{b, c, d, f\}$. Then the outline of the algorithm is as follows.

- A0. Given probe interval graph $G = (P, N, E)$, compute the enhanced probe interval graph $G^+ = (P, N, E \cup E^+)$;
- A1. Partition N into two subsets N^* and N_S;
- A2. Construct the $\mathcal{MPQ}$-tree T^* of $G^* = (P, N^*, E^*)$, where E^* is the set of edges induced by $P \cup N^*$ from G^+;
- A3. Embed each nonprobe v in N_S into T^*.

Note that the tree constructed in step A2 is an *ordinary* $\mathcal{MPQ}$-tree. In step A3, it will be modified to the extended $\mathcal{MPQ}$-tree. The following observation is obtained by definition:

Observation 4 Let v be a nonprobe in N_S. Then for any two vertices $u_1, u_2 \in N_{G^+}(v)$, $I_{u_1} \cap I_{u_2} \neq \emptyset$.

It is also easy to see that if $G = (P, N, E)$ is connected, so is G^*.

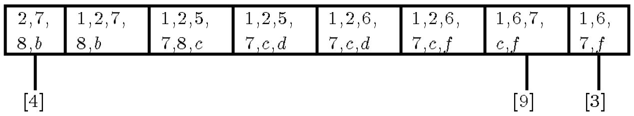

Figure 5: The canonical $\mathcal{MPQ}$-tree T^* of G^*

3.1 Construction of $\mathcal{MPQ}$-tree of G^*

Let $G^* = (P, N^*, E^*)$ be the enhanced probe interval graph induced by P and N^*. The following lemma plays an important role in this subsection.

Lemma 5 Let u and v be any nonprobes in N^*. Then there is an interval representation of G^* such that $I_u \cap I_v \neq \emptyset$ if and only if $\{u, v\} \in E^+$.

Proof. If $\{u, v\} \in E^+$, $I_u \cap I_v \neq \emptyset$ by definition. Thus we assume that $\{u, v\} \notin E^+$, and show that there is an interval representation of G^* such that $I_u \cap I_v = \emptyset$. We fix an interval representation of G^*, and assume that $I_u \cap I_v \neq \emptyset$. When $N(u) \cap N(v) = \emptyset$, it is easy to modify to satisfy $I_u \cap I_v = \emptyset$. Thus we assume that $N(u) \cap N(v) \neq \emptyset$. We first show that $N(u) \not\subseteq N(v)$ and $N(v) \not\subseteq N(u)$. If $N(u) \subseteq N(v)$, since $\{u, v\} \notin E^+$, all vertices in $N(u)$ intersect with each other. Thus, $N(u)$ induces a clique, which contradicts $u \in N^*$. Hence $N(u) \not\subseteq N(v)$ and $N(v) \not\subseteq N(u)$. Without loss of generality, we can assume that $L(u) < L(v) < R(u) < R(v)$. Let w_1 and w_2 are any probes which intersect the interval $[L(v), R(u)]$. Then, since $\{u, v\} \notin E^+$, $I_{w_1} \cap I_{w_2} \neq \emptyset$. Thus, by the Helly property (see, e.g.,[1]), there is a point p in the interval $[L(v), R(u)]$ such that all probes contain p. We replace the point p in all intervals by a small interval $[p - \epsilon, p + \epsilon]$, and then we replace I_u by $[L(u), p - \epsilon]$ and I_v by $[p + \epsilon, R(v)]$. The replacement has no effect to the relations between u (or v) and probes. We here show that the replacement also has no effect to the relations between u (or v) and other nonprobes. To derive contradictions, we assume hat the relation between u and a nonprobe w is changed. Since the interval I_u is shortened, $w \in N(u)$ becomes $w \notin N(u)$ by the replacement. Since both of u and w are nonprobes, there are two independent probes t_1 and t_2 that guarantee $w \in N(u)$. Then, replacing $[L(u), R(u)]$ by $[L(u), p - \epsilon]$, at least one of t_1 and t_2, say t, changes from $t \in N(u)$ to $t \notin N(u)$. However this contradicts the definition of the point p, which should be contained in t, and we have $t \in N(u)$ after replacement. Thus the replacement has no effect to the relations between u (or v) and other nonprobes. Hence we obtain a new valid interval representation of G^* with $I_u \cap I_v = \emptyset$. Repeating this process for each pair we have the lemma. ∎

The definition of (enhanced) probe interval graphs and Lemma 5 imply the main theorem in this section:

Theorem 6 The enhanced probe interval graph $G^* = (P, N^*, E^*)$ is an interval graph.

Hereafter we call the graph $G^* = (P, N^*, E^*)$ *the backbone interval graph* of $G^+ = (P, N, E \cup E^+)$. For any given interval graph, its corresponding $\mathcal{MPQ}$-tree can be computed in linear time [12]. Thus we also have the following corollary:

Corollary 7 The canonical $\mathcal{MPQ}$-tree T^* of G^* can be computed in linear time.

Thus the step A2 is rewritten as follows;

A2. Construct the canonical $\mathcal{MPQ}$-tree T^* of the backbone interval graph $G^* = (P, N^*, E^*)$ of $G^+ = (P, N, E \cup E^+)$;

In the canonical $\mathcal{MPQ}$-tree T^*, for each pair of nonprobes u and v, their corresponding intervals intersect if and only if $\{u, v\} \in E^+$. This implies the following observation.

Observation 8 The canonical $\mathcal{MPQ}$-tree T^* gives us the possible interval representations of G^* such that two nonprobes in N^* do not intersect as possible as they can.

For example, for the graph $G = (P, N, E)$ in Figure 1, the canonical $\mathcal{MPQ}$-tree of the backbone interval graph $G^* = (P, N^*, E^*)$ is described in Figure 5. In the $\mathcal{MPQ}$-tree, $I_d \cap I_f = \emptyset$.

We note that in T^*, with suitable labels, we can distinguish nonprobes from probes, and nonprobes in N^* from nonprobes in N_S, which will be added later. Now, our main task is that embedding each vertex in N_S into the canonical $\mathcal{MPQ}$-tree T^* without breaking canonicality.

3.2 Embedding of Nonprobes in N_S

We first show two lemmas for the nonprobes in N_S.

Lemma 9 For each nonprobe v in N_S, all vertices in $N(v)$ are probes.

Proof. To derive a contradiction, we assume that a nonprobe v' is in $N(v)$. Then $\{v, v'\}$ is in E^+. Thus there are two probes u and u' such that $\{u, v\}$, $\{u, v'\}$, $\{u', v\}$, and $\{u', v'\}$ are in E, and $\{u, u'\}$ is not in E, which contradicts Observation 4. ∎

Lemma 10 For any probe interval graph G, there is an affirmative interval graph G' such that every nonprobe v in N_S of G is also simplicial in G'.

Proof. Let v be any nonprobe in N_S such that v is not simplicial in G'. By the Helly property, there is a point p such that all probes in $N(v)$ contains p. We replace the point p in all intervals by a small interval $[p-\epsilon, p+\epsilon]$, and we set $R(v) = L(v) = p$. Then v is simplicial in the interval graph corresponding to the new interval representation, and the interval graph is still affirmative. Thus, repeating this process, we have the lemma. ∎

By Lemma 10 and Theorem 2(d), we have the following corollary.

Corollary 11 For any probe interval graph G, there is an affirmative interval graph G' such that every nonprobe v in N_S of G is in a leaf of the $\mathcal{MPQ}$-tree of G'.

Our embedding is an extension of the embedding by Korte and Möhring [12] to deal with nonprobes. Each node $\hat{N}$ (including $\mathcal{Q}$-node) of the current tree T^* and each section S of a $\mathcal{Q}$-node is labeled according to how the nonprobe v in N_S is related to the probes in $\hat{N}$ or S. Nonprobes in $\hat{N}$ or S are ignored. The label is ∞, 1, or 0 if v is adjacent to all, some, or no probe from $\hat{N}$, or S, respectively. Empty sets (or the sets containing only nonprobes) obtain the label 0. Labels 1 and ∞ are called *positive* labels.

Lemma 12 For a nonprobe v in N_S, all nodes with a positive label are contained in a unique path of T^*.

Proof. By definition, v is simplicial, or $N(v)$ induce a clique. Thus Theorem 2(b) implies the lemma. ∎

Let P' be the unique minimal path in T^* containing all nodes with positive label. Let P be a path from the root of the $\mathcal{MPQ}$-tree T^* to a leaf containing P' (a leaf is chosen in any way). Let $\hat{N}_*$ be the lowest node in P with positive label. (That is, $\hat{N}_*$ is the node of the largest depth in P'.) If P contains nonempty $\mathcal{P}$-nodes or sections above $\hat{N}_*$ with label 0 or 1, let $\hat{N}^*$ be the highest such $\mathcal{P}$-node or $\mathcal{Q}$-node containing the section. Otherwise put $\hat{N}_* = \hat{N}^*$.

When $\hat{N}_* \neq \hat{N}^*$, we have the following lemma:

Lemma 13 We assume that $\hat{N}_* \neq \hat{N}^*$. Let $\hat{Q}$ be any $\mathcal{Q}$-node with sections $S_1, \cdots, S_k$ in this order between $\hat{N}_*$ and $\hat{N}^*$. If $\hat{Q}$ is not $\hat{N}^*$, all neighbors of v in $\hat{Q}$ appear in either S_1 or S_k.

Proof. We first observe that $\hat{N}^*$ contains at least one probe w of v with $w \notin N(v)$ since $\hat{N}^*$ is non-empty and the label of $\hat{N}^*$ is 0 or 1. We assume that v has a neighbor u in Q with $u \notin S_1$ and $u \notin S_k$ to derive a contradiction. Let U_1 and U_k be the set of vertices occurring below S_1 and S_k, respectively. By Lemma 3(c), $U_1 \neq \emptyset$ and $U_k \neq \emptyset$. Thus there are two vertices $u_1 \in U_1$ and $u_k \in U_k$ such that $I_{u_1} \subseteq I_w$, $I_u \subseteq I_w$, $I_{u_k} \subseteq I_w$, and $R(I_{u_1}) < L(I_u) < R(I_u) < L(u_k)$ (or $R(I_{u_k}) < L(I_u) < R(I_u) < L(u_1)$). Thus we have $I_u \subset I_w$, which contradicts that $w \notin N(v)$ and $u \in N(v)$. ∎

Note that Lemmas 12 and 13 correspond to [12, Lemma 4.1]. However, Lemma 13 does not hold at the node $\hat{N}^*$. We are now ready to use the bottom-up strategy from $\hat{N}_*$ to $\hat{N}^*$ as in [12]. In [12], the ordering of vertices are determined by LexBFS. In our algorithm, the step A3 consists of the following substeps;

A3.1. while there is a nonprobe v such that $\hat{N}_* \neq \hat{N}^*$ for v, embed v into T^*;

A3.2. while there is a nonprobe v such that $\hat{N}_* = \hat{N}^*$ for v and v is not a floating leaf, embed v into T^*;

A3.3. embed each nonprobe v (such that $\hat{N}_* = \hat{N}^*$ for v and v is a floating leaf) into T^*.

As shown later, an embedding of a nonprobe v with $\hat{N}_* \neq \hat{N}^*$ merges some nodes into one new $\mathcal{Q}$-node. Thus, during step A3.1, embedding of a nonprobe v can change the condition of other nonprobes u from "$\hat{N}_* \neq \hat{N}^*$" to "$\hat{N}_* = \hat{N}^*$". We note that A3.1 and A3.2 do not generate floating leaves, and all floating leaves are embedded

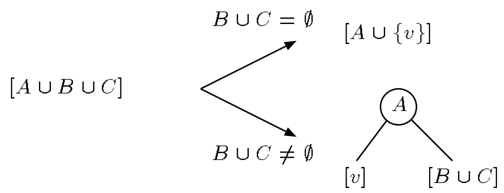

Figure 6: Template L1 when $\hat{N}^* = \hat{N}_*$ is a leaf

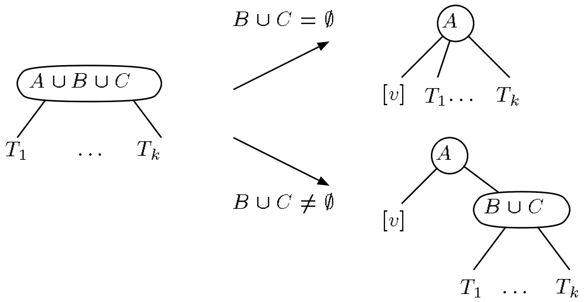

Figure 7: Template P1 when $\hat{N}^* = \hat{N}_*$ is a $\mathcal{P}$-node

in step A3.3, which will be shown later. Hence the templates used in steps A3.1 and A3.2 are not required to manage floating leaves.

Hereafter, we suppose that the algorithm picks up some nonprobe v from N_S and it is going to embed v into T^*. In most cases, the vertex set V_N of the current node or section is partitioned into A, B, and C defined as follows;

$$\begin{aligned} A &:= P \cap V_N \cap N(v), \\ B &:= (P \cap V_N) \setminus A, \\ C &:= N \cap V_N. \end{aligned}$$

Since we extend the templates in [12], we use the same names of templates as L1, P2, and so on, which is an extension of the corresponding templates in [12] (templates from Q4 to Q7 are new templates). We also use the help templates H1 and H2 in [12] if they can be applied; it is simple and omitted here. Through the embedding, we keep the following assertion:

Assertion 14 (1) Each nonprobe in N_S has no intersection with unnecessary nonprobes,

(2) each leaf contains either vertices in $P \cup N^*$ or one nonprobe in N_S, and

(3) each nonprobe in N_S is in a leaf.

3.2.1 Templates for the nonprobe with $\hat{N}_* = \hat{N}^*$

We first assume that $\hat{N}^* = \hat{N}_*$, which occurs in steps A3.2 and A3.3. If the node is a leaf or a $\mathcal{P}$-node, we use template L1 in Figure 6 or P1 in Figure 7, respectively. If $\hat{N}^* = \hat{N}_*$ is a $\mathcal{Q}$-node with sections $S_1, \cdots, S_k$ in this order, v can be a floating leaf. We let $A := (\cup_{1 \le i \le k} S_i) \cap N(v)$. Let ℓ be the minimum index with $A \subseteq S_\ell$ and r be the maximum index with $A \subseteq S_r$. That is, $A \not\subseteq S_i$ for each $i < \ell$ and $i > r$, and $A \subseteq S_j$ for each $\ell \le j \le r$. Then there are four cases:
(a) $\ell = 1$ and $A \subset S_\ell \cap P$. In the case, v may be a leaf of a new section $S_0 := A \subset S_1$. The case $r = k$ and $A \subset S_k \cap P$ is symmetric.
(b) $A = S_j \cap P$ for some $\ell \le j \le r$. In the case, v may be a leaf under the section S_j.
(c) $A = S_j \cap S_{j+1} \cap P$ for some $\ell \le j < r$. In the case, v may be a leaf under the new section S between S_j and S_{j+1} with $S := A \cup (S_j \cap S_{j+1} \cap N)$.

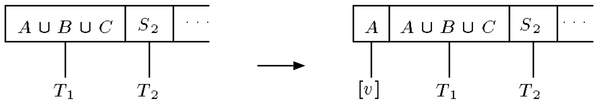

Figure 8: Template Q2 for (1) $\hat{N}_* = \hat{N}^*$ and $A \subset S_1 \cap P$, or (2) $\hat{N} = \hat{N}_* \neq \hat{N}^*$, $A \subseteq S_1$, and $A \not\subseteq \cap_{1 \le i \le k} S_i$

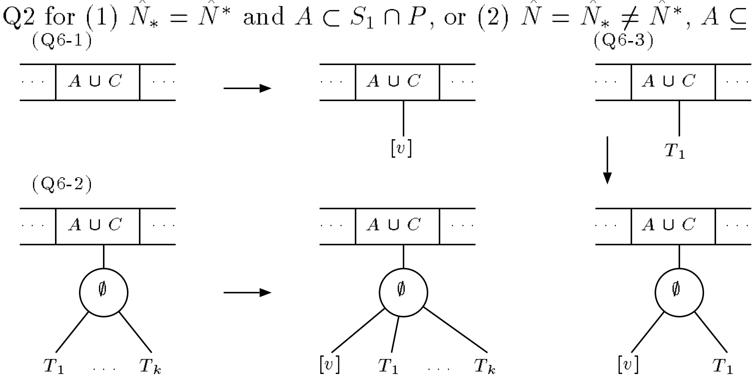

Figure 9: Template Q6 for $\hat{N}_* = \hat{N}^*$ and $A = S_j \cap P$ for some $\ell \le j \le r$

(d) $S_j \cap S_{j+1} \cap P \subset A \subset S_j \cap P$ or $S_j \cap S_{j+1} \cap P \subset A \subset S_{j+1} \cap P$ for some $\ell \le j < r$. In the case, v may be a leaf under the new section S between S_j and S_{j+1} with $S := A \cup (S_j \cap S_{j+1} \cap N)$.

The algorithm checks if the position of the v is uniquely determined. If it is uniquely determined, the algorithm embeds v into the place in step A3.2. If exactly one of the cases (a) to (d) occurs, we use the templates as follows. In the case (a), template Q2 in Figure 8 is used. In the case (b), we use three templates Q6-1, Q6-2, and Q6-3 in Figure 9 as follows; if the section S_j has no child, template Q6-1 is used and v is added as a leaf under S_j; if the root of the subtree under S_j is a $\mathcal{P}$-node with empty label, template Q6-2 is used and v is added as a leaf under the $\mathcal{P}$-node; or otherwise, template Q6-3 is used and v is added as a leaf under a new $\mathcal{P}$-node with empty label under S_j. We note that Assertion 14(2) holds if $\hat{R}$ contains nonprobes. In the case (c) or (d), template Q7 in Figure 8 is used; we note that we have $A \cup (S_j \cap S_{j+1} \cap N) = S_j \cap S_{j+1}$ in the case (c). We have one more case that the position of the v may be uniquely determined; $\ell = 1$, $r = k$, and $(S_i \cap S_{i+1} \cap P) \setminus A \neq \emptyset$ for each $1 \le i < k$. In the case, we use the template Q1-1 in Figure 11. In Figure 11, for each $1 \le i \le k$, $B_i := (S_i \cap P) \setminus N(v)$ and C_i denotes nonprobes in S_i. We note that $B_i \cap B_{i+1} \neq \emptyset$ for each $1 \le i < k$; otherwise, v can be a floating leaf under the section between S_i and S_{i+1}.

If the position is not uniquely determined, v is a floating leaf. Thus, in the case, the embedding is postponed until step A3.3. Then we use template Q4 in Figure 12 for such ℓ and r; in the figure, R_i denotes the set of floating leaves in S_i. Hereafter, we assume that each section S_i between R_ℓ and R_r knows if the vertex v can be a floating leaf "under S_i", "right of S_i", and "left of S_i"; that is, if v can be a leaf $[v]$ under S between S_i and S_{i+1}, they know that v can be a floating leaf at the left side of S_i, and the right side of S_{i+1}, respectively. If v can be a floating leaf under such sections (including non-existent sections), we say v can *hang down* the section.

We have the following observation.

Observation 15 In steps A3.2 and A3.3, all $\mathcal{Q}$-nodes are neither divided nor merged.

3.2.2 Templates for the nonprobe with $\hat{N}_* \neq \hat{N}^*$

When $\hat{N}_* \neq \hat{N}^*$, we use the same bottom-up strategy from $\hat{N}_*$ to $\hat{N}^*$ as in [12]. Let $\hat{N}$ denote the current node that starts from $\hat{N}_*$ and ends up at $\hat{N}^*$. The algorithm consists of three phases; (1) $\hat{N} = \hat{N}_*$, (2) $\hat{N} \neq \hat{N}_*$

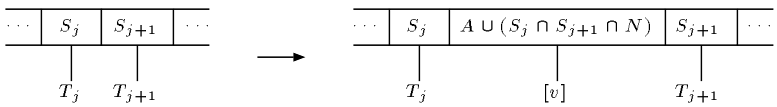

Figure 10: Template Q7 for $\hat{N}_* = \hat{N}^*$ and $S_j \cap S_{j+1} \cap P \subset A \subseteq S_{j+1} \cap P$ or $S_j \cap S_{j+1} \cap P \subset A \subseteq S_j \cap P$

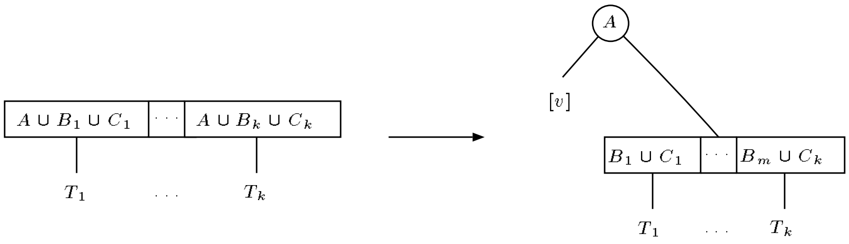

Figure 11: Template Q1-1

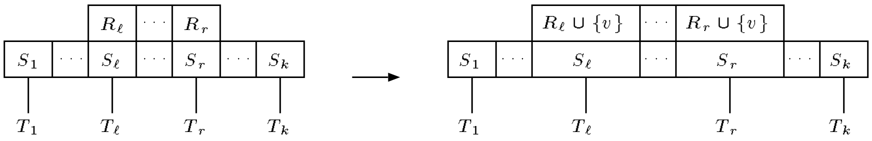

Figure 12: Template Q4

and $\hat{N} \neq \hat{N}^*$, and (3) $\hat{N} = \hat{N}^*$. The first two phases are the natural extension of the templates in [12] by Lemmas 12 and 13 which correspond to [12, Lemma 4.1]. However, the algorithm uses one more template in the third phase, since Lemma 13 does not hold. The templates in the case $\hat{N}_* \neq \hat{N}^*$ never generate floating leaves. Therefore, since they are applied in step A3.1, the templates in the case are not required to manage floating leaves.

(1) $\hat{N} = \hat{N}_* \neq \hat{N}^*$. Since the label of $\hat{N} = \hat{N}_*$ is positive, $A := \hat{N} \cap N(v) \neq \emptyset$. If $\hat{N}$ is a leaf or a $\mathcal{P}$-node, the algorithm uses template L2 in Figure 13 or P2 in Figure 14, respectively. When $\hat{N}$ is a $\mathcal{Q}$-node, we can use Lemmas 12 and 13 in this case. Thus we have two subcases, which correspond to templates Q1 and Q2 in [12]. By Lemma 13, we assume that $A \subseteq S_1$ without loss of generality. The algorithm uses template Q1-2 in Figure 15 if $A \subseteq S_k$, and otherwise, it uses template Q2 in Figure 8.

Observation 16 In any case, v becomes a leaf $[v]$ under a non-empty section S_1 of a $\mathcal{Q}$-node since $A \neq \emptyset$.

(2) $\hat{N} \neq \hat{N}_*$ and $\hat{N} \neq \hat{N}^*$. If $\hat{N}$ is a $\mathcal{P}$-node, the algorithm uses template P3 in Figure 16. If $\hat{N}$ is a $\mathcal{Q}$-node, we can use Lemmas 12 and 13 again and the algorithm uses template Q3 in Figure 17. By a simple induction of the length of the path P with Observation 16, we again have the following observation (since $S_1 \neq \emptyset$ in Figures 16 and 17):

Observation 17 In any case, v becomes a leaf $[v]$ under a non-empty section S_1 of a $\mathcal{Q}$-node.

(3) $\hat{N} = \hat{N}^* \neq \hat{N}_*$. If $\hat{N}$ is a $\mathcal{P}$-node, the algorithm uses the template P3 in Figure 16 again. If $\hat{N}$ is a $\mathcal{Q}$-node, we cannot use Lemmas 13. Let S_i' be the section in $\hat{N}$ such that the subtree T_i' contains $[v]$. If S_i' is the leftmost

Figure 13: Template L2 for $\hat{N} = \hat{N}_* \neq \hat{N}^*$

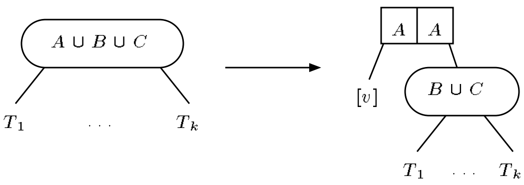

Figure 14: Template P2 for $\hat{N} = \hat{N}_* \neq \hat{N}^*$

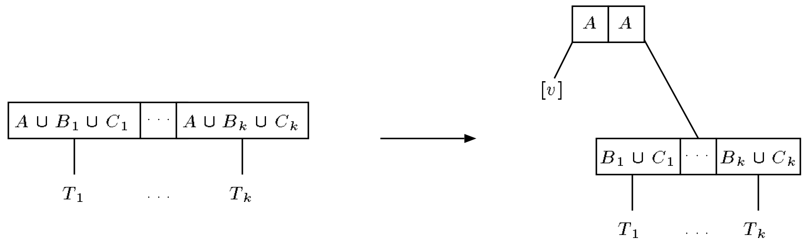

Figure 15: Template Q1-2 for $\hat{N} = \hat{N}_* \neq \hat{N}^*$ and $A \subseteq \cap_{1 \leq i \leq k} S_i$

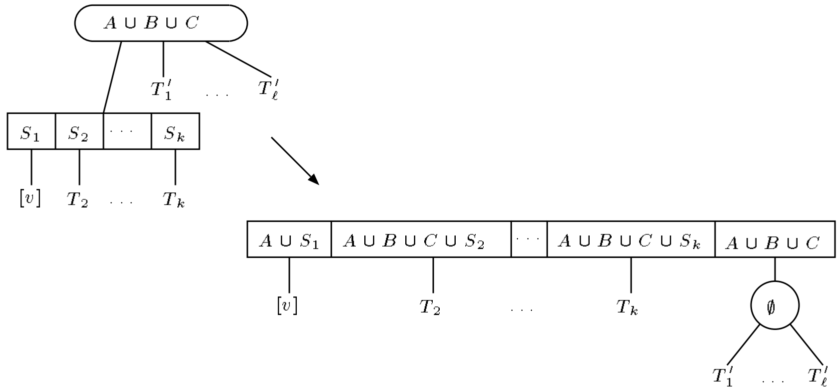

Figure 16: Template P3 for $\hat{N} \neq \hat{N}_*$ and $\hat{N} \neq \hat{N}^*$

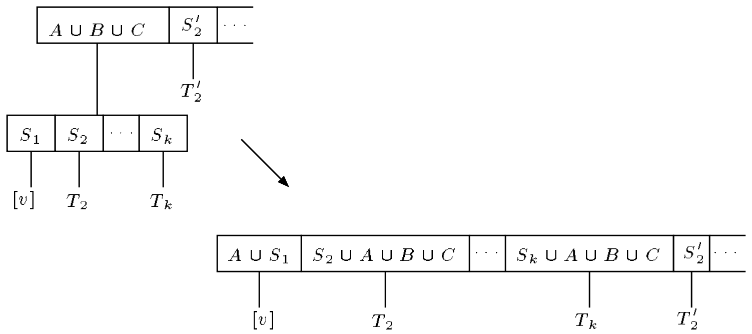

Figure 17: Template Q3 for $\hat{N} \neq \hat{N}_*$ and $\hat{N} \neq \hat{N}^*$

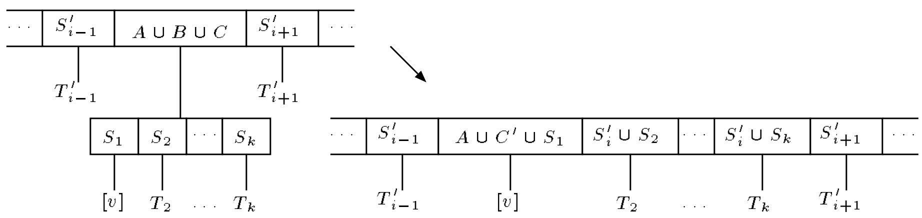

Figure 18: Template Q5 for $\hat{N} = \hat{N}^* \neq \hat{N}_*$ and $B \subseteq S'_{i+1}$

or rightmost section in $\hat{N}$, we can use the template Q3 in Figure 17 again. Thus we assume that $1 < i < k'$, where k' is the number of sections in the $\mathcal{Q}$-node $\hat{N}$. Let S'_{i-1} and S'_{i+1} be the left and right sections of S'_i, respectively. We now define $A := N(v) \cap S'_i$ and $B := (S'_i \cap P) \setminus A$. Then, since the label of S'_i is 0 or 1, we have $B \neq \emptyset$. For the set B, we have the following lemma:

Lemma 18 Either $B \subseteq S'_{i+1} \setminus S'_{i-1}$ or $B \subseteq S'_{i-1} \setminus S'_{i+1}$.

Proof. Let u be any vertex in B. By theorem 2(c), $u \in S'_{i-1}$ or $u \in S'_{i+1}$. Since $B \neq \emptyset$, $S'_{i-1} \cap B = S'_{i+1} \cap B = \emptyset$ does not occur. Thus it is sufficient to show that $S'_{i-1} \cap B \neq \emptyset$ and $S'_{i+1} \cap B \neq \emptyset$ implies a contradiction. We assume that there are two vertices u_1 and u_2 such that $u_1 \in S'_{i-1} \cap B$ and $u_2 \in S'_{i+1} \cap B$. Then $[v]$ cannot be a leaf under any sections S'_{i-1}, S'_i, and S'_{i+1}. This implies that the graph G is not a probe interval graph, which is a contradiction. ∎

Without loss of generality, we assume that Lemma 18(a) occurs. That is, all vertices in B appear from the section S'_i to the some sections on the right side of S'_i. Let $C' := S'_{i-1} \cap S'_i \cap N$. That is, C' is the set of nonprobes appearing both of S'_{i-1} and S_i. Then we use template Q5 in Figure 18. In the figure, C denotes the nonprobes in S'_i; that is, $S'_i = A \cup B \cup C$. We note that $C' \subseteq C$, and Assertion 14(1) holds.

Example 19 For the graph $G = (P, N, E)$ in Figure 1 with its backbone interval graph in Figure 5, the extended $\mathcal{MPQ}$-tree $\tilde{T}$ is shown in Figure 4. The algorithm uses templates L2 and Q3 to embed a, and uses template Q4 to embed g since it is a floating leaf. For the nonprobe e, only the case (c) in Section 3.2.1 can be applied; $\{1, 2, 7, 8, c, d\} \cap \{1, 2, 6, 7, c, d\} \cap P = \{1, 2, 7\} = N(e)$. Thus its position is uniquely determined, and embedded between the sections. Note that we can know that e intersects both of c and d with neither experiments nor enhanced edges. We also note that I_a and I_b could have intersection, but they are standardized according to Assertion 14(1).

3.3 Analysis of Algorithm

Correctness

Since the correctness of steps A0, A1, and A2 follows from Theorem 6, we concentrate on step A3. First, the templates cover all formally distinct cases. All templates for the case $\hat{N}_* = \hat{N}^*$ with the help-templates H1 and H2 in [12] are easily shown to be correct. Thus we consider the case $\hat{N}_* \neq \hat{N}^*$.

Theorem 20 When $\hat{N}_* \neq \hat{N}^*$, v is not a floating leaf.

Proof. We first assume that $\hat{N} = \hat{N}^* \neq \hat{N}_*$. Let S_1 be the section having the leaf $[v]$. If the algorithm uses the template P3 or Q3, the same technique in the proof of Theorem 4.3 in [12] works: Since the label of $\hat{N}$ is 0 or 1, $B \neq \emptyset$ in the templates, and S_1 is not empty by Observation 17. Thus there are no other place that $[v]$ can be put into. Thus we assume that the algorithm uses the template Q5. Since S_1 is not empty by Observation 17, $[v]$ cannot be under the sections in $\hat{N}$ except S'_i. On the other hand, since $B \neq \emptyset$, $[v]$ cannot be under the sections $A \cup B \cup C \cup S'_1$, $A \cup B \cup C \cup S'_2$, $\cdots$, $A \cup B \cup C \cup S'_k$. Thus, the only possible sections are between S'_{i-1} and S'_i or between S'_i and S'_{i+1}. However, by Lemma 18, one of them is prohibited. Thus the place of $[v]$ is uniquely determined, and v is not a floating leaf. ∎

We have the following corollary which corresponds to Corollary 4.4 in [12]:

Corollary 21 When $\hat{N}_* \neq \hat{N}^*$, all nodes properly between $\hat{N}_*$ and $\hat{N}^*$ on the path P will become inner sections of a $\mathcal{Q}$-node after embedding of v.

Theorem 22 The resulting extended $\mathcal{MPQ}$-tree is canonical up to isomorphism.

Proof. To derive contradictions, given probe interval graph G, assume that we have two nonisomorphic trees T_1 and T_2 for G. We moreover suppose that G has the minimum number of vertices among such graphs. Then there are two vertices v_1 in T_1 and v_2 in T_2 such that both of v_1 and v_2 correspond to v in G, and v_1 and v_2 guarantee that T_1 is not isomorphic to T_2. When v is in G^*, we immediately have a contradiction to Corollary 7. Thus, v is a simplicial nonprobe in N_S. If the positions of v_1 and v_2 are uniquely determined when they are embedded, we can show that v_1 and v_2 have to be embedded in the same place using the same argument in [13], which derives a contradiction. Thus at least one of v_1 and v_2 is a floating leaf and embedded in step A3.3. If both of v_1 and v_2 are floating leaves embedded in step A3.3, by Observation 15, the positions of v_1 and v_2 are the same place since $N(v_1) = N(v_2)$. Thus, without loss of generality, we assume that v_1 is a floating leaf embedded in A3.3, and v_2 is embedded in A3.1 or A3.2. However, by Theorem 20, v_2 is not a floating leaf, which is a contradiction. ∎

Implementation and Complexity

Theorem 23 For given probe interval graph $G = (P, N, E)$, let $\tilde{T}$ be the canonical extended $\mathcal{MPQ}$-tree, and $G^+ = (P, N, E \cup E^+)$ be the corresponding enhanced interval graph. Let $\tilde{E}$ be the set of edges $\{v_1, v_2\}$ joining nonprobes v_1 and v_2 which is given by $\tilde{T}$; more precisely, we regard $\tilde{T}$ as an ordinary $\mathcal{MPQ}$-tree, and the graph $\tilde{G} = (P \cup N, E \cup E^+ \cup \tilde{E})$ is the interval graph given by the $\mathcal{MPQ}$-tree $\tilde{T}$ (thus a floating leaf is not a leaf; the vertex appears in consecutive sections in the corresponding $\mathcal{Q}$-node). Then $\tilde{T}$ can be computed in $O((|P| + |N|)|E| + |E^+| + \left|\tilde{E}\right|)$ time and $O(|P| + |N| + |E| + |E^+| + \left|\tilde{E}\right|)$ space.

Proof. Let Δ be the maximum degree of probes in $G = (P, N, E)$. Then the step A0 can be performed in $O(|P| + |N| + \Delta|E|)$ time and $O(|P| + |N| + |E| + |E^+|)$ space shown in Appendix A. An enhanced probe interval graph is chordal graph [18, 20]. Thus, using Lemma 1, all simplicial nonprobes can be found in linear time and space. Thus the step A1 can be performed in linear time and space. The step A2 can be performed in $O(|P| + |N^*| + |E^*|)$ time and space. The implementation of step A3 is based on the algorithms in [3, 12]. When the vertex v is not a floating leaf, the algorithm deals with the new set C with the set B, and the additional process requires $O(1)$ time and space per iteration with modified implementations in [3, 12]. Thus, its total running time and space for such vertices would be $O(|P \cup N| + \left|E \cup E^+ \cup \tilde{E}\right|)$ in amortized manner (see [3, 12] for further details). When the vertex v is a floating leaf, the algorithm checks if the one of the conditions (a), (b), (c), (d) in Section 3.2.1 holds for the section S_i for each i with $1 \le i \le k$. Each check at section S_i can be performed in $O(deg(v))$ time, and $k = O(|P| + |N|)$ in general. Thus this step requires $O((|P| + |N|)deg(v))$ time for each nonprobe in N_S. Hence the total running time required to deal with floating leaves through the algorithm is bounded above by $O((|P| + |N|)|E|)$. Therefore we have the theorem. ∎

Corollary 24 The graph isomorphism problem for the class of (enhanced) probe interval graphs G is solvable in $O(n^2 + nm)$ time and $O(n^2)$ space, where n and m are the number of vertices and edges of an affirmative interval graph of G, respectively.

Proof. By Theorem 22, given (enhanced) probe interval graphs G_1 and G_2, $G_1 \sim G_2$ if and only if their corresponding canonical extended $\mathcal{MPQ}$-trees are isomorphic. If they have no floating leaves, using the same technique in [13, 6], the graph isomorphism problem can be solved in linear time. Even if they have floating leaves, it is not difficult to see that comparing two floating leaves v in G_1 and G_2 can be done in $O(deg(v))$ time, which completes the proof. ∎

We note that $|E^+| + \left|\tilde{E}\right|$ can be $\Theta(|N|^2) = \Theta(n^2)$ even if $|E| = O(n)$, where $n = |P| + |N|$. Thus the running time in the main theorem can be $\Theta(n^3)$ even if given probe interval graph has $O(n)$ edges.

4 Applications

Given canonical extended $\mathcal{MPQ}$-tree $\tilde{T}$, using a standard depth first search technique, we can compute in linear time if each subtree in $\tilde{T}$ contains only nonprobes. Thus, hereafter, we assume that each section S_i knows if its subtree contains only nonprobes or not.

4.1 Relations between nonprobes

We first consider the following problem:

Input: An enhanced probe interval graph $G^+ = (P, N, E \cup E^+)$ and the canonical extended $\mathcal{MPQ}$-tree $\tilde{T}$;

Output: Mapping f from each pair of nonprobes u, v with $\{u, v\} \notin E^+$ to "intersecting", "potentially intersecting", or "independent";

We denote by E_i and E_p the sets of the pairs of intersecting nonprobes, and the pairs of potentially intersecting nonprobes, respectively. That is, each pair of nonprobes u, v is either in E^+, E_i, E_p, or otherwise, they are independent.

Theorem 25 The sets E_i and E_p can be computed in $O(|E| + |E^+| + |E_i| + |E_p|)$ time for given enhanced probe interval graph $G^+ = (P, N, E \cup E^+)$ and the extended $\mathcal{MPQ}$-tree $\tilde{T}$.

Proof. We first analyze the relation between two nonprobes. Let u and v be two nonprobes such that $dep(u) \leq dep(v)$, u is in node $\hat{N}_u$, and v is in node $\hat{N}_v$. When $\hat{N}_u$ (and $\hat{N}_v$) is a $\mathcal{Q}$-node, we assume that u appears from S_{ul} to S_{ur} (and v appears from S_{vl} to S_{vr}, respectively). There are three cases we have to consider:
(1) $\hat{N}_u = \hat{N}_v$. If $\hat{N}_u (= \hat{N}_v)$ is a leaf, by Assertion 14(2), both of u and v are in N^*. Moreover, by Theorem 2(d), u and v are simplicial. Thus u and v are in N_S, which is a contradiction. Thus $\hat{N}_u$ is not a leaf. If $\hat{N}_u$ is a $\mathcal{P}$-node, it has at least two subtrees T_1 and T_2 under $\hat{N}_u$. If both of T_1 and T_2 contain probes, we have $\{u, v\} \in E^+$, which contradicts that $\{u, v\} \notin E^+$. Thus at most one subtree T_1 under $\hat{N}_u$ contains probes, and all probes in T_1 intersect with each other. We first assume that such subtree T_1 containing probes exists. Then, by the construction of the canonical extended $\mathcal{MPQ}$-tree, u and v should be added after probes in T_1 and they should be embedded as leaves under T_1, which is a contradiction. Thus all subtrees of $\hat{N}_u$ contain only nonprobes, which also contradicts Assertion 14(1). Therefore $\hat{N}_u$ is not a $\mathcal{P}$-node. If $\hat{N}_u$ is a $\mathcal{Q}$-node, we have three subcases.
(1-1) Neither u nor v are floating leaves. In the case, $\{u, v\} \in E_i$ if and only if they share common sections; that is, $\max\{ul, vl\} \leq \min\{ur, vr\}$. If $\max\{ul, vl\} - 1 = \min\{ur, vr\}$, $\{u, v\} \in E_p$; otherwise, u and v are independent.
(1-2) u is a floating leaf, and v is not (symmetric case is omitted). Let $\mathcal{S}_u$ be the set of sections that u can hang down, and $\mathcal{S}_v$ be the set of sections between S_{vl} and S_{vr}. If $\mathcal{S}_u \subseteq \mathcal{S}_v$, we have $\{u, v\} \in E_i$. If $\mathcal{S}_u \cap \mathcal{S}_v \neq \emptyset$ and $\mathcal{S}_u \setminus \mathcal{S}_v \neq \emptyset$, we have $\{u, v\} \in E_p$. Otherwise ($\mathcal{S}_u \cap \mathcal{S}_v = \emptyset$), they are independent.
(1-3) Both of u and v are floating leaves. Let $\mathcal{S}_u$ and $\mathcal{S}_v$ be the sets of sections that u and v can hang down, respectively. In the case, $\{u, v\} \in E_p$ if and only if $\mathcal{S}_u \cap \mathcal{S}_v \neq \emptyset$. Otherwise, they are independent.
(2) $\hat{N}_u$ is an ancestor of $\hat{N}_v$ (symmetric case is omitted). If $\hat{N}_u$ is a $\mathcal{P}$-node, clearly, $\{u, v\} \in E_i$. Thus we assume that $\hat{N}_u$ is a $\mathcal{Q}$-node, and $\hat{N}_v$ is in the subtree T_i under the section S_i in $\hat{N}_u$.

We first assume that u is not a floating leaf. Then, $\{u, v\} \in E_i$ if and only if $ul \leq i \leq ur$. If $i < ul - 1$ or $i > ur + 1$, they are independent. We assume that $i = ul - 1$ or $i = ur + 1$. In the case, intuitively, if the vertex v can appear in the leftmost or rightmost node in T_i, $\{u, v\}$ is in E_p, or otherwise, they are independent. To check this, we use the following procedure $\mathsf{Adj}(\hat{N}_u, v)$, and we have $\{u, v\} \in E_p$ if it returns "Yes", or otherwise they are independent:

Procedure $\mathsf{Adj}(\hat{N}_u, v)$:

(1) for each node $\hat{N}$ on the path joining the nodes $\hat{N}_u, \cdots, \hat{N}_v$, do the following;

- (1.1) if $\hat{N} = \hat{N}_u$, proceed to the next node;
- (1.2) if $\hat{N}$ is a $\mathcal{P}$-node and $\hat{N} \neq \hat{N}_v$, proceed to the next node;
- (1.3) if $\hat{N}$ is a $\mathcal{Q}$-node and $\hat{N} \neq \hat{N}_v$, do the following;
  - (1.3.1) let S_i be the section in $\hat{N}$ such that its subtree contains $\hat{N}_v$, and let k be the number of sections in $\hat{N}$;
  - (1.3.2) if all sections $S_1, S_2, \cdots, S_{i-1}$ have subtrees containing only nonprobes, and all probes in sections $S_1, S_2, \cdots, S_{i-1}$ are contained in S_i, or its symmetric case holds, proceed to the next node; otherwise, return "No";
- (1.4) if $\hat{N} = \hat{N}_v$, do the following;

(1.4.1) if $\hat{N}$ is a leaf or a $\mathcal{P}$-node, return "Yes";

(1.4.2) if $\hat{N}$ is a $\mathcal{Q}$-node with sections $S_1, \cdots, S_k$, do the following;

(1.4.2.1) let S_i be the section in $\hat{N}$ that contains $L(v)$, and S_j be the section in $\hat{N}$ that contains $R(v)$;

(1.4.2.2) if $i = 1$ or $j = k$ then return "Yes";

(1.4.2.3) if all sections $S_1, S_2, \cdots, S_{i-1}$ have subtrees containing only nonprobes, and all probes in sections S_1, S_2, $\cdots$, S_{i-1} are contained in S_i, or its symmetric case holds, return "Yes", otherwise, return "No".

We next assume that u is a floating leaf. If u cannot hang down S_{i-1}, S_i, S_{i+1}, the section between S_{i-1} and S_i, and the section between S_i and S_{i+1}, clearly, u and v are independent. Thus we assume that u can hang down at least one of those sections. If the subtree of S_i contains only nonprobes, by Assertion 14(1) and (2), $\{u, v\} \in E_p$. Thus, we assume that the subtree of S_i contains not only nonprobes, but also probes w. In the case, $w \notin N(u)$. Thus we have $\{u, v\} \in E_p$ if and only if there are arrangements of intervals such that the interval $[\min\{R(u), R(v)\}, \max\{L(u), L(v)\}]$ contains no $R(w)$s and $L(w)$s for each probe w in the subtree. More precisely, $\{u, v\} \in E_p$ if and only if $\mathsf{Adj}(\hat{N}_u, v)$ returns "Yes", or otherwise, u and v are independent.

(3) $\hat{N}_u$ is not an ancestor of $\hat{N}_v$, and $\hat{N}_v$ is not an ancestor of $\hat{N}_u$. Let $\hat{N}_c$ be the nearest common ancestor of $\hat{N}_v$ and $\hat{N}_u$. In the case, $\{u, v\} \in E_p$ if and only if there are arrangements of intervals such that the interval $[\min\{R(u), R(v)\}, \max\{L(u), L(v)\}]$ contains no $R(w)$s and $L(w)$s for each probe w in the subtree rooted at $\hat{N}_c$. Thus we can determine if $\{u, v\} \in E_p$ using the following algorithm:

(1) If $\hat{N}_c$ is a $\mathcal{P}$-node, $\{u, v\} \in E_p$ if and only if both of $\mathsf{Adj}(\hat{N}_c, u)$ and $\mathsf{Adj}(\hat{N}_c, v)$ return "Yes", otherwise, u and v are independent.

(2) If $\hat{N}_c$ is a $\mathcal{Q}$-node, let S_i and S_j be the sections in $\hat{N}_c$ such that S_i and S_j contain $\hat{N}_v$ and $\hat{N}_u$, respectively. Without loss of generality, we assume that $i < j$. Then, $\{u, v\} \in E_p$ if and only if both of $\mathsf{Adj}(\hat{N}_c, u)$ and $\mathsf{Adj}(\hat{N}_c, v)$ return "Yes", and either (a) $j - i = 1$ or (b) all sections $S_{i+1}, \cdots, S_{j-1}$ have subtrees containing only nonprobes, and $S_i \cap P = S_{i+1} \cap P = \cdots = S_{j-1} \cap P = S_j \cap P$. Otherwise, u and v are independent.

Now we prove the theorem. The correctness of the above analysis can be done by the induction for the length of the path(s) between $\hat{N}_u$ and $\hat{N}_v$; which is straightforward but the details are rather tedious, and therefore omitted. Using the standard dynamic programming technique from the leaves to the root, those relations can be computed in $O(\,|E| + |E^+| + |E_i| + |E_p|\,)$ time and space. ∎

By Theorem 25, we can heuristically find the "best" nonprobe to fix the structure of the DNA sequence:

Corollary 26 For given enhanced probe interval graph $G^+ = (P, N, E \cup E^+)$ and the canonical extended $\mathcal{MPQ}$-tree $\tilde{T}$, we can find the nonprobe v that has most potentially intersecting nonprobes in $O(\,|E| + |E^+| + |E_i| + |E_p|\,)$ time.

4.2 Enumeration of all affirmative interval representations

We next consider the following problem:

Input: A probe interval graph $G = (P, N, E)$ and the canonical extended $\mathcal{MPQ}$-tree $\tilde{T}$;

Output: All affirmative interval graphs.

Theorem 27 For given enhanced probe interval graph $G = (P, N, E)$ and the canonical extended $\mathcal{MPQ}$-tree $\tilde{T}$, all affirmative interval graphs can be enumerated in polynomial time and space of $|P| + |N| + |M|$, where M is the number of the affirmative interval graphs.

Proof. We here show how to generate one possible affirmative interval graph of G. It is easy to modify it to enumerate all affirmative interval graphs in polynomial time and space of $|P| + |N| + |M|$. We first fix each floating leaf as a leaf under the corresponding $\mathcal{Q}$-node (in arbitrary way). Then, we have an affirmative $\mathcal{MPQ}$-tree for some interval graph. However, to generate all possible interval graphs, we have to consider two more cases; (1) two adjacent nonprobes in N^* might have intersection as noted in Observation 8, and (2) two adjacent nonprobes in N_S might have intersection as noted in Assertion 14(1). Those two cases can be analyzed in the same case-analysis in the proof of Theorem 25. Then we next fix the relations between each pair of nonprobes (We note that some pair of nonprobes u and v may be determined by the relation of the other pair of nonprobes u and w). It is easy to see that for each possible affirmative interval graph, its $\mathcal{MPQ}$-tree can be generated in this way. ∎

5 Concluding Remarks

It may seem to be straightforward to modify the algorithm to solve the recognition problem for probe interval graphs (with its vertex partition). However, it is not true. Our algorithm does not mind the consistency of the floating leaves. In section 3.2.1, let us suppose there are many floating leaves v_i that satisfy the condition (d); $S_j \cap S_{j+1} \cap P \subset A_i \subset S_j \cap P$ for some fixed j and $i = 1, 2, \cdots$. In the case, we have to check if they can be linearly sorted in inclusion at this point. The check of the consistency can be solved in $O(n^2)$ time. Thus it is possible to modify our algorithm to solve the recognition problem for the (enhanced) probe interval graphs; but the algorithms in [11, 14, 10] are faster.

References

[1] C. Berge. *Hypergraphs*. Elsevier, 1989.

[2] J.R.S. Blair and B. Peyton. An Introduction to Chordal Graphs and Clique Trees. In *Graph Theory and Sparse Matrix Computation*, volume 56 of *IMA*, pages 1–29. (Ed. A. George and J.R. Gilbert and J.W.H. Liu), Springer, 1993.

[3] K.S. Booth and G.S. Lueker. Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using *PQ*-Tree Algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

[4] A. Brandstädt, F.F. Dragan, H.-O. Le, V.B. Le, and R. Uehara. Tree Spanners for Bipartite Graphs and Probe Interval Graphs. In *29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '03)*, pages 106–118. Lecture Notes in Computer Science Vol. 2880, Springer-Verlag, 2003.

[5] A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. SIAM, 1999.

[6] C.J. Colbourn and K.S. Booth. Linear Time Automorphism Algorithms for Trees, Interval Graphs, and Planar Graphs. *SIAM Journal on Computing*, 10(1):203–225, 1981.

[7] P. Galinier, M.Habib, and C. Paul. Chordal Graphs and Their Clique Graphs. In *WG '95*, pages 358–371. Lecture Notes in Computer Science Vol. 1017, Springer-Verlag, 1995.

[8] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.

[9] W.-L. Hsu and T.-H. Ma. Substitution Decomposition on Chordal Graphs and Applications. In *ISA '91*, pages 52–60. Lecture Notes in Computer Science Vol. 557, Springer-Verlag, 1991.

[10] J.L. Johnson, R.M. McConnell, and J.P. Spinrad. Linear Time Recognition of Probe Interval Graphs. in preparation, 2002.

[11] J.L. Johnson and J.P. Spinrad. A Polynomial Time Recognition Algorithm for Probe Interval Graphs. In *Proc. 12th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 477–486. ACM, 2001.

[12] N. Korte and R.H. Möhring. An Incremental Linear-Time Algorithm for Recognizing Interval Graphs. *SIAM Journal on Computing*, 18(1):68–81, 1989.

[13] G.S. Lueker and K.S. Booth. A Linear Time Algorithm for Deciding Interval Graph Isomorphism. *Journal of the ACM*, 26(2):183–195, 1979.

[14] R.M. McConnell and J.P. Spinrad. Construction of Probe Interval Models. In *Proc. 13th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 866–875. ACM, 2002.

[15] T.A. McKee and F.R. McMorris. *Topics in Intersection Graph Theory*. SIAM, 1999.

[16] F.R. McMorris, C. Wang, and P. Zhang. On Probe Interval Graphs. *Discrete Applied Mathematics*, 88:315–324, 1998.

[17] T. Nagoya, R. Uehara, and S. Toda. Completeness of Graph Isomorphism Problem for Bipartite Graph Classes. In *IEICE Technical Report*, volume COMP2001-93, pages 1–5, 3/12 2002.

[18] P. Zhang. Probe Interval Graphs and Its Applications to Physical Mapping of DNA. manuscript, 1994.

[19] P. Zhang. Probe Interval Graph and Its Applications to Physical Mapping of DNA. RECOMB 2000, Poster Session; available at `http://recomb2000.ims.u-tokyo.ac.jp/Posters/list-posters.html`, 2000.

[20] P. Zhang. United States Patent. Method of Mapping DNA Fragments. [Online] Available `http://www.cc.columbia.edu/cu/cie/techlists/patents/5667970.htm`, July 3 2000.

A Computing Enhanced Edges

In this section, we show an efficient algorithm for generating enhanced edges. More precisely, given probe interval graph $G = (P, N, E)$, the set E^+ of enhanced edges can be computed in $O(|P| + |N| + \Delta |E|)$ time and $O(|P| + |N| + |E| + |E^+|)$ space, where Δ is the maximum degree of probes in P. We note that $|E^+|$ can be $\Theta((|P| + |N|)^2)$ even if $|E| = O(|P| + |N|)$ in general. The algorithm contains two phases.

First, the algorithm constructs one of possible interval representations $\mathcal{I} = \{I_1, \cdots, I_{n_1}\}$ of the interval graph $G[P]$, which runs in $O(|P| + |E|)$ time and space (see, e.g., [12]). Without loss of generality, we assume that the intervals do not share common endpoints, and each endpoint is an integer in $[1, 2|P|]$. We note that $G[P]$ is not connected in general, while $G = (P, N, E)$ is connected.

Then the second phase of the algorithm is the following:

C0. set $E^+ := \emptyset$;

C1. for each $i = 1, 2, \cdots, 2|P|$ do the following;

- C1.1. if $i = R(I_v)$ for some I_v, take each pair of nonprobes $\{u_1, u_2\}$ with $u_1, u_2 \in N(v)$, and record "the pair $\{u_1, u_2\}$ is a candidate";
- C1.2. if $i = L(I_v)$ for some I_v, take each pair of nonprobes $\{u_1, u_2\}$ with $u_1, u_2 \in N(v)$, and if the pair $\{u_1, u_2\}$ is a candidate, add it into E^+;

Theorem 28 The algorithm finds all enhanced edges in $O(|P| + |N| + \Delta |E|)$ time and $O(|P| + |N| + |E| + |E^+|)$ space.

Proof. We first show the correctness. Let u and u' be nonprobes with $\{u, u'\} \in E^+$. Then there are two independent probes v and v' such that $I_v \cap I_u \neq \emptyset$, $I_{v'} \cap I_u \neq \emptyset$, $I_v \cap I_{u'} \neq \emptyset$, and $I_{v'} \cap I_{u'} \neq \emptyset$. Then, in any fixed interval representations of G, v and v' are independent. Without loss of generality, we assume that $R(v) < L(v')$ in the interval representation. Then, when $i = R(v)$ in step C1.1, the algorithm stores $\{u, u'\}$ as a candidate, and when $i = L(v')$, the algorithm adds $\{u, u'\}$ into E^+. It is easy to see that the edges added into E^+ are all enhanced edges, which completes the proof of the correctness of the algorithm.

Next we analyze complexity. In the implementation, in step C1.1, each vertex u in $N(v)$ keeps "u is marked by v as candidates". In step C1.2, the algorithm first collects all vertices u in $N(v)$, and check the set of vertices marked by a common vertex v' for each v' in $N(u)$. It is easy to check the algorithm requires $O(|P| + |N| + |E| + |E^+|)$ space. For each i, step C1.1 runs in $O(deg(v))$ time, and step C1.2 runs in $O(\sum_{u \in N(v)} deg(u))$. Thus the total running time of the algorithm is $\sum_{v \in P}(O(deg(v)) + O(\sum_{u \in N(v)} deg(u))) = O(|P| + |N| + \Delta |E|)$ time. ∎