

A Test for Interval Graphs on Noisy Data
– DNA Fragment Assembly

Wei-Fu Lu

Institute of Computer and Information Science
National Chiao Tung University, Hsin-chu, Taiwan, ROC

email: gis84812@cis.nctu.edu.tw

TEL: 886-2-7883799 EXT. 1602

Wen-Lian Hsu*

Institute of Information Science
Academia Sinica, Taipei, Taiwan, ROC

email: hsu@iis.sinica.edu.tw

TEL: 886-2-7883799 EXT. 1804

FAX: 886-2-7824814

* The corresponding author.

ABSTRACT

An interval graph is the intersection graph of a collection of intervals. One important application of interval graph is the construction of physical maps in genome research, that is, to reassemble the clones to determine the relative position of fragments of DNA along the genome. The linear time algorithm by Booth and Lueker (1976) for this problem has a serious drawback: the data must be error-free. However, laboratory work is never flawless. We devised a new iterative clustering algorithm based on local structure matching, which is robust enough to accommodate a certain percentage of noisy data and to produce a likely interval model realizing the original graph.

Key Word: interval graph recognition, physical mapping, DNA sequence assembly, probe hybridization, clustering algorithm.

1. INTRODUCTION

A graph is an *interval graph* if it is the intersection graph of a collection of intervals on a straight line. This class of graphs has a wide range of applications. An important application of interval graphs is the construction of physical maps for the genome research. Physical maps are critical in hunting for specific genes of interest, and also useful for further physical examination of DNA required for other genome project. The term “physical mapping” means the determination of the relative position of fragments of DNA along the genome by physicochemical and biochemical methods. The construction of physical maps is generally accomplished as follows. Long DNA sequences are broken to smaller fragments, and then each fragment is reproduced into the so-called *clones*. After deciding some fingerprints for each clone, two clones are considered overlapping if their fingerprints are sufficiently similar. Finally, information on pairwise overlapping determines the relative positions of clones, thus completing the construction of physical maps [OHCD1985], [CB1989], [CS1989], [GC1987], [OD1989], [GO1990], [MCZ1987].

The error free version of the mapping problem can be modeled as an interval graph recognition problem: given a graph $G=(V,E)$, finding a family of intervals such that each interval corresponding to one vertex of the graph, and two vertices are adjacent if and only if their corresponding intervals are overlapping [BL1976], [KM89], [HM1991], [Hsu 1992]. However, data collected from laboratories unavoidably contain errors, such as *false positives* (FPs, two overlapping clones are actually non-overlapping) and *false negatives* (FNs, two non-overlapping clones are actually overlapping). Traditional recognition algorithms can hardly be applied on noisy data directly, because a single error might cause the clone assembly to fail. Moreover, no straightforward extension of traditional algorithm can overcome the drawbacks.

Four typical models have been proposed for dealing with errors. The definitions are as follows: 1) *interval graph completion problem*: assume the input data only contain FNs and minimize the number of edges whose addition makes the graph an interval [KS1996, KST1999, NS1998]. 2) *interval graph deletion problem*: assume there are only FPs in the input data, and minimize the number of edges whose deletion makes the graph an interval graph [GGKS1995]. 3) *interval sandwich problem*: assume that some pairs of clones are definite overlaps, some are definite non-overlaps,

and the rest are unknown, then construct an interval graph under these overlapping constraints [GKS1994], [KS1999]. 4) *intervalizing k -color graph problem*: assume that clones are created from k copies of DNA molecule, and some pairs of clones are definite overlaps. The objective is to generate a k -colorable interval graph with the overlapping conditions [GKS1994], [GGKS1995], [FHW1993], [BF1996]. However, the above models suffer from the following two unpleasant phenomena: 1. all of the above models have been shown to be NP-hard [Y1981], [FHW1993], [GKS1994], [GGKS1995], and it would be difficult to define an associated “single objective optimization problem” for approximation since the errors could be intertwined together; 2. even if one can find a perfect solution under certain restrictions, such a solution might not make any biological sense.

To cope with this dilemma, consider the nature of error treatment. Generally, data collected in real life contain a small percentage of errors. Suppose the error percentage is 5% with careful control. The challenge is thus to discover the 95% correct information versus the 5% incorrect information automatically. We designed an algorithm to deal with errors based on local structure matching. The idea is very similar to the one employed in [Hsu 2003]. Our philosophy is that, in order to determine whether certain overlapping information is valid or noisy, we check the neighborhood data to see if it conforms “approximately” to a particular local structure dictated by the problem. The probability that an isolated piece of spurious information has a well-behaved neighborhood structure is nil. More precisely, in our analysis, if there is enough valid information in the input data, then a certain monotone structure of the overlapping information on the neighborhood will emerge, allowing us to weed out most errors. We do not set any “global” objective to optimize. Rather, our algorithm tries to maintain (or restore) such a “local” monotone structure as much as we can. In doing that, it is sometimes advantageous to “delete” noisy intervals, namely, those that corrupt the monotone structure.

The kind of error-tolerant behavior considered here are similar in nature to algorithms for voice recognition or character recognition problems. Thus, it would be difficult to “guarantee” that the clustering algorithm always produces a desirable solution (such as one that is a fixed percentage away from the so-called “optimal solution”); the result should be justified through benchmark data and real life experiences. Our experimental results show that, when the error percentage is small, our clustering algorithm is robust enough to discover certain errors and to correct

them automatically most of the time.

The remaining sections are organized as follows. Section 2 gives the basic definitions of some notations. An interval graph test based on [LH2003] is discussed in Section 3, which forms the basis of our clustering algorithm. Section 4, the main part of this paper, illustrates how to deal with errors in the input data. The experimental results are shown in Section 5. Section 6 contains some concluding remarks.

2. BASIC DEFINITIONS

In this section, we give definitions and terminologies that we will need in the sequel. For other graph-theoretical definitions see [G1980].

All graphs are assumed to be undirected, simple, and finite in this paper. For a graph $G = (V, E)$, denote its number of vertices by n and its number of edges by m . Given a vertex u in G , define $N[u]$ to be the set of vertices including u and those vertices adjacent to u in G ; define $N(u)$ to be $N[u] - \{u\}$. For some subset M of V , define $N(M)$ be the set of those vertices that are not in M but adjacent to some vertices in M . Thus, we have $N(N[u]) = \{x \mid x \text{ is not in } N[u] \text{ but adjacent to some vertices in } N[u]\}$, which is the second-tier neighborhood in a breadth-first-search from u . This kind of neighborhood plays a crucial role on our clustering analysis. We define relations between two adjacent vertices using the above set of neighbors. Two adjacent vertices u, v in G are said to be *strictly adjacent (STA)*, if none of $N[u]$ and $N[v]$ is contained in the other. We denote the set of vertices strictly adjacent to u by $STA(u)$. A vertex u is said to *be contained in* another vertex v , if $N[u]$ is contained in $N[v]$.

Each interval graph has a corresponding interval model in which two intervals overlap if and only if their corresponding vertices are adjacent[§]. However, the corresponding interval model is usually far from unique, because of variations of the endpoint orderings. To obtain the unique interval model representation, consider the following block structure of endpoints: Denote the right (resp. left) endpoint of an interval u by $R(u)$ (resp. $L(u)$). In an interval model, define a maximal contiguous set of right (resp. left) endpoints as an *R-block* (resp. *L-block*). Thus, the endpoints can be grouped as an alternating left-right block sequence. Since an endpoint block is a set, the endpoint orderings within the block are ignored. The overlapping relationship remains unchanged if one permutes the endpoint order within each block. Denote the right block containing $R(u)$ by $B_R(u)$, the left block containing $L(u)$ by $B_L(u)$, and the set of block subsequence from $B_L(u)$ to $B_R(u)$ by $[B_L(u), B_R(u)]$. An endpoint $R(w)$

[§] For convenience, we shall not distinguish between these two terms, “vertex” and its corresponding “interval”.

(resp. $L(w)$) is said to be *contained in an interval u* if $B_L(w)$ (resp. $B_R(w)$) is contained in $[B_L(u), B_R(u)]$.

Let G be an interval graph. Consider an interval model for G . For an interval u , the neighborhood of u can be partitioned into $A(u)$, $B(u)$, $C(u)$ and $D(u)$ as follows:

- $A(u) = \{ w \mid w \text{ strictly overlaps } u \text{ from the left side} \}$
- $B(u) = \{ w \mid w \text{ strictly overlaps } u \text{ from the right side} \}$
- $C(u) = \{ w \mid w \text{ properly contains } u \}$
- $D(u) = \{ w \mid w \text{ is properly contained in } u \}$

We call these sets $A(u)$, $B(u)$, $C(u)$, $D(u)$, the left neighborhood, the right neighborhood, the outer neighborhood, and the inner neighborhood. Furthermore, the second-tier neighborhood of u can be partitioned into $LL(u)$ and $RR(u)$ as follows:

- $LL(u) = \{ w \mid w \text{ is completely to the left of } u \text{ and overlaps some neighbors of } u \}$
- $RR(u) = \{ w \mid w \text{ is completely to the right of } u \text{ and overlaps some neighbors of } u \}$

We call $LL(u)$ the left second-tier neighborhood and $RR(u)$ the right second-tier neighborhood. An example of $A(u)$, $B(u)$, $C(u)$, $D(u)$, $LL(u)$, and $RR(u)$ is shown in Figure 2.1.

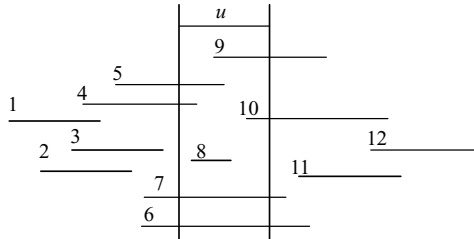


Figure 2.1. An example of $A(u)$, $B(u)$, $C(u)$, $D(u)$, $LL(u)$, and $RR(u)$, where $A(u)=\{4,5\}$, $B(u) =\{9,10\}$, $C(u) =\{6,7\}$, $D(u) =\{8\}$, $LL(u) =\{1,2,3\}$, and $RR(u) =\{11,12\}$.

Note that the above sets can be easily derived if an interval model is given. It is more difficult to derive them if only the edge adjacency is known. Such a neighborhood classification is a main step in our test.

3. AN INTERVAL GRAPH TEST

To our best knowledge, no straightforward extension of existing linear time algorithms can handle errors. The idea of [LH2003], however, can be modified to yield a clustering version that can deal with noisy data. In this section, we describe a quadratic time interval graph test, which adopts some techniques similar to [LH2003]. This algorithm will be modified to deal with noisy data in Section 4. Notably, the time complexity is not a major concern for algorithms on noisy data.

The basic idea of this algorithm is very simple: The vertices are processed one by one according to an ascending order of their degree. For each vertex u , we decide the unique left-right block sequence that records the relative positions of endpoints within u , based on a robust local structure on its neighbors. If the unique left-right block sequence within u intersects other existing left-right block sequences, all the left-right block sequences are further merged into a new left-right block sequence. Finally, if graph G is an interval graph, after all vertices have been processed, we will obtain the unique left-right block sequence that realize graph G ; otherwise, the algorithm will terminate in some iteration due to the failure of left-right block sequence construction.

For each vertex u in G , our algorithm performs three main steps: 1) *neighborhood classification*, 2) *block sequence determination*, and 3) *vertex replacement*. The first step, neighborhood classification, classifies vertices adjacent to u into $A(u)$, $B(u)$, $C(u)$ and $D(u)$. Since the block sequence within u relates to a robust local structure on $A(u)$ and $B(u)$, this classification is significant for our interval graph test. The second step, block sequence determination, decides the unique left-right block sequence within u according to a monotone structure on $A(u)$ and $B(u)$, and merge this block sequence with another existing block sequence, if necessary. The last step, vertex replacement, generates a "special vertex" u^s which is adjacent to all neighbors of u and special vertices strictly adjacent to u . We shall associate u^s with the corresponding left-right block sequence of u constructed in the second step. Remove vertices whose endpoints are both contained in the block sequence of u^s , and delete all edges between $A(u^s)$ and $B(u^s)$, since information about those deleted edges and vertices is no longer needed. After vertex replacement, the graph is further reduced.

The main iteration of our interval graph test is described in Figure 3.1, and an example is shown in Figure 3.2. The following definitions are needed to describe the algorithm.

Definition 3.1 A collection of sets is said to be *monotone* if every two sets S_i, S_j in the collection are comparable, that is, either $S_i \supseteq S_j$ or $S_j \supseteq S_i$.

Definition 3.2 An interval u is said to be *compatible* with a left-right block sequence $LB_1, RB_1, LB_2, RB_2, \dots, LB_d, RB_d$ if

- (1) the left (resp. right) endpoints within u are contained in $LB_1 \cup LB_2 \cup \dots \cup LB_d$ (resp. $RB_1 \cup RB_2 \cup \dots \cup RB_d$), and
- (2) let RB_{j_1} (resp. LB_{j_2}) be the leftmost R -block (resp. rightmost L -block) having nonempty intersection with endpoints within u . Then all blocks in between (but excluding) RB_{j_1} and LB_{j_2} are contained in $N(u)$.

The *Interval-graph-test*: Process an original vertex u .

- 1 Neighborhood Classification:
 - 1.1 Construct the following set: $C(u) \leftarrow \{ w \mid N(w) \supset N(u) \}$, $D(u) \leftarrow \{ w \mid N(w) \subseteq N(u) \}$ and $STA(u) \leftarrow N(u) - C(u) - D(u)$.
 - 1.2 Partition $STA(u)$ into $A(u)$ and $B(u)$:
 - (1) Let u^* be a vertex in $STA(u)$ with the largest $|N(u^*) \cap (N(STA(u)) - N(u))|$.
 - (2) Let $LL(u) \leftarrow \{ w \mid w \in N(u^*) \cap (N(STA(u)) - N(u)) \}$,
and $RR(u) \leftarrow N(STA(u)) - N(u) - LL(u)$.
 - (3) Let $A(u) \leftarrow STA(u) \cap N(LL(u))$ and $B(u) \leftarrow STA(u) - A(u)$.
 - 1.3 Let u_{SL} be the special interval, if any, in $A(u)$, and u_{SR} the special interval, if any, in $B(u)$.
- 2 Block sequence determination:
 - 2.1 Find the collection of sets $\{ N(w) \cap B(u) \mid w \in A(u) \}$.
 - 2.2 Check the following:
 - (1) The collection $\{ N(w) \cap B(u) \mid w \in A(u) \}$ is monotone such that the right endpoints of intervals in $A(u)$ and the left endpoints of intervals in $B(u)$ can be uniquely partitioned with $R(u_{SL})$ located in the first R -block and $L(u_{SR})$ located in the last L -block.
 - (2) Every interval in $D(u)$ is compatible with the block sequence determined by the above two sets and the remaining intervals in $D(u)$.
 - 2.3 If there is any violation, G is not an interval graph and the test is terminated
- 3 Vertex replacement:
 - 3.1 Create new special interval u^s with $N(u^s) \leftarrow N(u_{SL}) \cup N(u) \cup N(u_{SR})$.
 - 3.2 Suppose that x is a vertex with its right endpoint in u^s but not its left endpoint, and y is a vertex with its left endpoint in u^s but not its right endpoint. Remove edge (x, y) if it exists.
 - 3.3 Remove u , u_{SL} and u_{SR} and vertices whose left endpoints and right endpoints are both contained in u^s .

Figure 3.1. The *Interval-graph-test*.

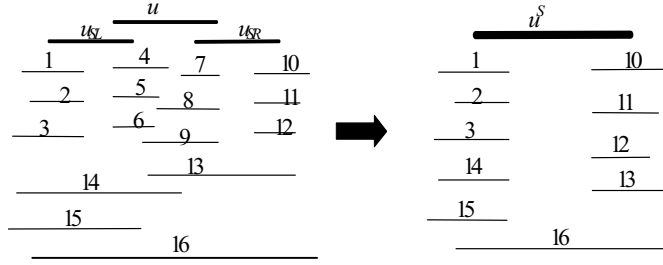


Figure 3.2. An example of the *Interval-graph-test*.

The left half of Figure 3.2 is the interval graph at the beginning of the iteration that interval u is processed. Intervals u_{SL} and u_{SR} are the two special intervals strictly overlapping u . The corresponding block sequence of u_{SL} is $\{L(u_{SL})\}$, $\{R(1)\}$, $\{L(4)$, $L(5)\}$, $\{R(2), R(3)\}$, $\{L(6), L(u)\}$, $\{R(u_{SL})\}$, and the corresponding block sequence of u_{SR} is $\{L(u_{SR})\}$, $\{R(7), R(8)\}$, $\{L(10), L(11)\}$, $\{R(9), R(u)\}$, $\{L(12)\}$, $\{R(u_{SR})\}$. In neighborhood classification, the neighborhood of u is classified into $A(u) = \{14, 15\}$, $B(u) = \{13\}$ and $C(u) = \{16\}$, and $D(u) = \{4, 5, 6, 7, 8, 9\}$. Based on the monotone structures of $A(u)$ and $B(u)$, compatible property of $D(u)$ and the block sequence decided by $A(u)$ and $B(u)$, we can obtain the block sequence within u , say $\{L(u)\}$, $\{R(u_{SL})\}$, $\{L(9), L(13)\}$, $\{R(6)\}$, $\{L(8)\}$, $\{R(4), R(5)\}$, $\{L(7)\}$, $\{R(14)\}$, $\{L(u_{SR})\}$, $\{R(u)\}$. In the vertex replacement step, do the following:

- (1) create a new special interval u^s with $N(u^s) = N(u_{SL}) \cup N(u) \cup N(u_{SR}) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$, and associates u^s with the block sequence that records the relative positions of endpoints within u^s .
- (2) delete all edges connecting vertices in $\{4, 5, 6, 14, 15\}$ and $\{7, 8, 9, 13\}$.
- (3) remove intervals u_{SL} , u_{SR} , and intervals contained in u^s (namely, intervals 4, 5, 6, 7, 8, and 9).

At the end of this iteration, the corresponding interval graph becomes the right half of Figure 3.2.

We shall prove in Theorem 3.6 that, for a graph G , our algorithm correctly decides whether G is an interval graph or not. Several lemmas are needed for the proof of Theorem 3.6.

Below, we shall adopt the notations used in the *Interval-graph-test* in Figure 3.1.

Lemma 3.3 *If graph G is an interval graph, then the collections $\{ N(w) \cap B(u) \mid w \in A(u) \}$ and $\{ N(w) \cap A(u) \mid w \in B(u) \}$ are both monotone.*

Proof. We shall prove that $\{ N(w) \cap B(u) \mid w \in A(u) \}$ is monotone. Because intervals in $A(u)$ overlap u from the left and intervals in $B(u)$ overlap u from the right, we have that right endpoints of intervals in $B(u)$ are to the right of right endpoints of intervals in $A(u)$. Let intervals w_1 and w_2 be two intervals in $A(u)$ such that $R(w_1)$ is to the left of $R(w_2)$. Each interval v in $B(u)$ adjacent to w_1 is also adjacent to w_2 , since $L(v)$ is to the left of $R(w_1)$ and $R(w_2)$. Thus $N(w_1) \cap B(u) \subseteq N(w_2) \cap B(u)$ and we have that $\{ N(w) \cap B(u) \mid w \in A(u) \}$ is monotone. The proof of that $\{ N(w) \cap A(u) \mid w \in B(u) \cup D_B(u) \}$ is monotone is symmetric. ■

Lemma 3.4 *If $\{ N(w) \cap B(u) \mid w \in A(u) \}$ and $\{ N(w) \cap A(u) \mid w \in B(u) \}$ are monotone, then the right endpoints in $A(u)$ and the left endpoints in $B(u)$ can be partitioned into LB_2, \dots, LB_n and $RB_1, RB_2, \dots, RB_{n-1}$ respectively, such that $LB_1, RB_1, \dots, LB_n, RB_n$ is the left-right block sequence within u , where $LB_1 = \{L(u)\}$ and $RB_n = \{R(u)\}$.*

Proof. Let $\{ N(w) \cap B(u) \mid w \in A(u) \}$ and $\{ N(w) \cap A(u) \mid w \in B(u) \}$ be monotone. Partition intervals in $A(u)$ into ordered set $RB_1, RB_2, \dots, RB_{n-1}$ such that for any two vertices w_1 and w_2 in $A(u)$, $N(w_1) \cap B(u) = N(w_2) \cap B(u)$, if w_1 and w_2 in the same set; and $N(w_1) \cap B(u) \subset N(w_2) \cap B(u)$, if $w_1 \in RB_i$ and $w_2 \in RB_j$ such that $i < j$. Similarly, partition intervals in $B(u)$ into ordered set LB_2, LB_3, \dots, LB_n such that for any two intervals w_1 and w_2 in $B(u)$, $N(w_1) \cap A(u) = N(w_2) \cap A(u)$, if w_1 and w_2 in the same set; and $N(w_1) \cap A(u) \supset N(w_2) \cap A(u)$, if $w_1 \in LB_i$ and $w_2 \in LB_j$ such that $i < j$. Let $LB_1 = \{L(u)\}$ and $RB_n = \{R(u)\}$. It is easy to check that $LB_1, RB_1, \dots, LB_n, RB_n$ is the left-right block sequence that induces the relative positions of the right endpoints of intervals in $A(u) \cup \{u\}$ and the left endpoints of intervals in $B(u) \cup \{u\}$. ■

Theorem 3.5 *A graph is an interval graph iff the following conditions hold at each iteration of the interval-graph-test:*

1. *The collections of sets $\{ N(w) \cap B(u) \mid w \in A(u) \}$ and $\{ N(w) \cap A(u) \mid w \in B(u) \}$ are monotone and the right endpoints of $A(u)$ and the left endpoints of $B(u)$ can be uniquely partitioned with $R(u_{SL})$ located on the first right block and $L(u_{SR})$ located on the last left block.*
2. *Every interval in $D(u)$ is compatible with the block sequence determined by the above two sets and the remaining intervals in $D(u)$.*

If the given graph G is an interval graph, then the proposed algorithm will yield an interval model of graph G , otherwise, the algorithm will terminate in step 2 of the Interval-graph-testing.

Proof. We first show the “only if” part. From lemma 3.3 one can easily check the necessity of these conditions for an interval graph. The reason that $R(u_{SL})$ should be located in the first right block and $L(u_{SR})$ should be located in the last left block could be argued as follows. Assume that $R(u_{SL})$ be not located in the first right block, there exists interval w such that $R(w)$ is in the first right block. Since the right endpoint of w is located to the right of $L(u)$ and to the left of $R(u_{SL})$, we have $w \in A(u_{SL})$ and $u \in B(u_{SL})$ and $w \in N(u)$ when u is being processed. This is a contradiction to that edge (w, u) should have been removed at the iteration that u_{SL} was processed, we conclude that $R(u_{SL})$ should be located in the first right block. Similarly, $L(u_{SR})$ should be located in the last left block. Note that condition 2 simply indicates that we can merge an existing special interval into our ongoing block sequence correctly.

Now, consider the “if” part, we shall use induction on $|E(G)|$. Assume the statement is true for graphs smaller than G . From lemma 3.4, if these conditions are satisfied at every iteration, then for each interval u processed, the main iteration of our algorithm could determines the unique left-right block partition of right endpoints of intervals in $A(u) \cup D_A(u) \cup \{u\}$ and the left endpoints of intervals in $B(u) \cup D_B(u) \cup \{u\}$. Further, refine this partition by bringing in compatible interval in $D_S(u)$ one by one. Let k be the first iteration that some edges are deleted in step (3.2) of the main iteration. Let u_k^S be the special vertex that is generated at the k -th iteration, and G' be the reduced graph. Since $|E(G')| < |E(G)|$ and the two conditions in Theorem 3.5 are satisfied at every iteration after iteration k , by the induction hypothesis, G' is an interval graph. Consider any interval model of interval graph G' , say $LB'_1, RB'_1, LB'_2, \dots, LB'_d, RB'_d$. Let the corresponding block sequence of u_k^S be $LB''_1, RB''_1, LB''_2, \dots, LB''_h, RB''_h$. Consider the following two cases.

Case 1. Only one endpoint of interval u_k^S is contained in $LB'_1, RB'_1, LB'_2, \dots, LB'_d, RB'_d$.

Without loss of generality, assume that $u_k^S \in A(u_t)$ for some $t > k$. We have that $LB''_1, RB''_1, LB''_2, \dots, LB''_h, RB'_1, LB'_2, \dots, LB'_d, RB'_d$ is the left-right block sequence that is the interval model realizing graph G .

Case 2. Both two endpoints of interval u_k^S are contained in $LB'_1, RB'_1, LB'_2, \dots, LB'_d, RB'_d$.

We have that $LB''_1, RB''_1, LB''_2, \dots, LB''_h, RB''_h$ is compatible with $LB'_1, RB'_1, LB'_2, \dots, LB'_d, RB'_d$. Thus, $RB'' \subseteq RB'_i$ and $LB'' \subseteq LB'_{i+1}$ for some i , where $RB'' = RB''_1 \cup RB''_2 \cup \dots \cup RB''_{h-1}$ and $LB'' = LB''_2 \cup LB''_3 \cup \dots \cup LB''_h$. Now, merge $LB''_1, RB''_1, LB''_2, \dots, LB''_h$ with $LB'_1, RB'_1, LB'_2, \dots, LB'_d$, we get a left-right block sequence $LB'_1, RB'_1, LB'_2, \dots, LB'_i, RB'_i - RB'', \{L(u_k)\}, RB'_i \cap RB'', LB'_{i+1} \cap LB'', \{R(u_k)\}, LB'_{i+1} - LB'', RB'_{i+1}, \dots, LB'_d, RB'_d$ that realizes the interval model for G . Therefore, these conditions are also sufficient. ■

4. TREATING ERRORS

In this section, we present a clustering version of the interval graph test. We shall consider FPs and FNs simultaneously. Assume the number of FPs is at most a quarter of that of FNs (which seems to be practical for most biological experiments). Such an assumption is important because FPs are much more troublesome than FNs. In these clustering algorithms, we need to set different threshold values to detect various errors. Whenever possible, we shall provide motivations for these threshold values by proving some lemmas for the more ideal situations. The method to perform neighbor classification on noisy data will be discussed in Section 4.1. Section 4.2 illustrates the block sequence determination while taking FNs and FPs into account. The complete clustering version of interval graph test (under noise) is summarized in Section 4.3.

4.1 *The error-tolerant neighborhood classification*

If the input data contain errors, it is more intricate for neighborhood classification. However, based on clustering analysis on the neighborhood and second-tier neighborhood of u , we are able to classify neighbors of interval u roughly into four sets $A(u)$, $B(u)$, $C(u)$, and $D(u)$. Our strategy is to classify the second-tier neighbors of u , $N(N[u])$, into $LL(u)$ and $RR(u)$ first, and then classify the neighbors of u into $A(u)$, $B(u)$, $C(u)$ and $D(u)$ based on $LL(u)$ and $RR(u)$. Let $OV(w,v) = |N[w] \cap N[v]|$ denote the *overlap function* between two intervals w and v . The overlap function is used to measure the degree of overlapping for each pair of intervals in $N(N[u])$. The clustering of $LL(u)$ and $RR(u)$ uses a greedy strategy based on the overlap function. The classification of $LL(u)$ and $RR(u)$ is described in Figure 4.1 below.

The *LL-RR-classification* Algorithm

1. For each interval in $N(N[u])$, associate it with a cluster consisting of that interval initially.
2. Calculate $OV(w,v)$ for each pair of intervals in $N(N[u])$.
3. Select a pair of intervals w and v , from two different clusters, attaining the highest $OV(u,v)$ value. Merge the corresponding clusters of w and v into one cluster.
4. Reiterate Step 3 until are two clusters left.
5. Let one cluster be $LL(u)$ and the other be $RR(u)$.

Figure 4.1. The *LL-RR-classification* Algorithm

We now classify $STA(u)$ into $A(u)$ and $B(u)$ based on the following heuristic rule: intervals in $A(u)$ should not overlap any interval in $RR(u)$, and intervals in $B(u)$ should

not overlap any interval in $LL(u)$. Thus, any overlapping relations between $A(u)$ and $RR(u)$, and those between $B(u)$ and $LL(u)$ could be considered as FPs. For each interval w in $SAT(u)$, classify w into $A(u)$ if the number of FPs created by classifying w into $A(u)$ is less than the number of FPs created by classifying w into $B(u)$. Otherwise, classify w into $B(u)$. Such a classification scheme is summarized in Figure 4.2.

The *A-B-classification* Algorithm

For each interval w in $STA(u)$:

1. Calculate the error functions of w as follows:

$$E_A(w) \leftarrow |\{(w, v) \mid v \in N(w) \cap RR(u)\}|$$

$$E_B(w) \leftarrow |\{(w, v) \mid v \in N(w) \cap LL(u)\}|.$$

2. Classify w into $A(u), B(u)$:

If $E_A(w) < E_B(w)$ then classify w into $A(u)$

else classify w into $B(u)$

Figure 4.2. The *A-B-classification* Algorithm.

The above sets $A(u)$, $B(u)$, $LL(u)$ and $RR(u)$ could still be misclassified due to those FPs and FNs related to interval u itself. To prevent this kind of errors (or to minimize its effect), we shall reclassify intervals currently in $LL(u) \cup A(u)$ into new $LL(u)$ and $A(u)$ as follows (The reclassification of $RR(u) \cup B(u)$ into new $RR(u)$ and $B(u)$ can be done similarly).

Denote $LL(u) \cup A(u)$ by $L\text{-part}(u)$, and $RR(u) \cup B(u)$ by $R\text{-part}(u)$. To reclassify intervals currently in $LL(u) \cup A(u)$ into new $LL(u)$ and $A(u)$, it suffices to determine the location of $L(u)$. Once $L(u)$ is located, then those intervals of $L\text{-part}(u)$ whose right endpoints are to the right (respectively, left) of $L(u)$ are considered neighbors, $A(u)$ (respectively, non-neighbors, $LL(u)$), of u . We shall locate the right endpoint of intervals in $L\text{-part}(u)$ first, and then decide the position of $L(u)$. To do that, we need to determine the relative positions among right endpoints of intervals in $L\text{-part}(u)$. Interestingly enough, $R\text{-part}(u)$ will play an important role in this process based on the following simple lemma.

Lemma 4.1.1 *Let S and T be two sets of intervals. If the right (respectively, left) endpoint of every interval in T is to the right (respectively, left) of the right (respectively, left) endpoint of every interval in S , then the right (respectively, left) endpoint of interval w in S with the largest $|N(w) \cap T|$ value is the rightmost right*

endpoint (respectively, leftmost left endpoint) among all right (respectively, left) endpoints of intervals in S .

Based on Lemma 4.1.1, we shall order right endpoints of intervals in $L\text{-part}(u)$ from right to left iteratively as follows. Initially, set S to be $L\text{-part}(u)$ and T to be $R\text{-part}(u)$. Note that S and T will be changed at each iteration. We shall maintain that the right endpoint of every interval in T is to the right of the right endpoints of intervals in S . Thus, we shall make the right endpoint of interval $w_{|L\text{-part}(u)|}$ in S with the largest $|N(w_{|L\text{-part}(u)|}) \cap T|$ value to be the rightmost right endpoint of intervals in S ($= L\text{-part}(u)$). Next, delete interval $w_{|L\text{-part}(u)|}$ from S and add $w_{|L\text{-part}(u)|}$ into T . Now, make the right endpoint of interval $w_{|L\text{-part}(u)|-1}$ in the resultant S with the largest $|N(w_2) \cap T|$ value to be the rightmost right endpoint of intervals in the remaining S ($= L\text{-part}(u) - \{w_{|L\text{-part}(u)|}\}$). Thus, $R(w_{|L\text{-part}(u)|-1})$ is to the left of $R(w_{|L\text{-part}(u)|})$, but to the right of all right endpoints of other intervals in $L\text{-part}(u)$. Then, delete interval $w_{|L\text{-part}(u)|-1}$ from S and add $w_{|L\text{-part}(u)|-1}$ into T . Reiterate the above process until all right endpoints of intervals in $L\text{-part}(u)$ have been ordered.

After that, call the ordered right endpoints of intervals in $L\text{-part}(u)$ from left to right as $R(w_1), R(w_2), \dots, R(w_{|L\text{-part}(u)|})$. However, in some cases due to noise, although $|N(x) \cap T| > |N(y) \cap T|$ (which would entail that $R(x)$ is to the right of $R(y)$), $R(x)$ might be, in fact, to the left of $R(y)$. However, if the error rate is quite small, (say, no more than 5%), we can expect that $R(x)$ will be ordered to the right of $R(y)$ with high probability. Thus, we can obtain the approximate ordering of the right endpoints of intervals in $L\text{-part}(u)$. Similarly, we can also locate left endpoints of intervals in $R\text{-part}(u)$ from left to right as $L(v_1), L(v_2), \dots, L(v_{|R\text{-part}(u)|})$.

To decide the position of $L(u)$, we calculate the ‘‘cost’’ of $L(u)$ for each position that $L(u)$ could be placed. If $L(u)$ is placed between $R(w_i)$ and $R(w_{i+1})$, u must overlap all intervals w_j with $j > i$, otherwise (u, w_j) is a FN. Moreover, intervals w_j and w_k such that $j, k > i$ must overlap each other, otherwise (w_j, w_k) is a FN. On the other hand, intervals w_j with $j \leq i$ should not overlap u , otherwise (w_i, u) is a FP. Let $ErrL(u, i)$ be the total number of FNs and FPs, if $L(u)$ is placed between $R(w_i)$ and $R(w_{i+1})$. Thus, $ErrL(u, i) = |\{(u, w_j) \mid (u, w_j) \notin E(G) \text{ and } j > i\}| + |\{(w_j, w_k) \mid (w_j, w_k) \notin E(G) \text{ and } j, k > i\}| + |\{(w_i, u) \mid (w_i, u) \in E(G) \text{ and } j \leq i\}|$. Note that $ErrL(u, 0)$ is defined as the total number of errors that place $L(u)$ to the left of all the right endpoints of intervals in $L\text{-part}(u)$. We conclude that $L(u)$ should be placed between $R(w_i)$ and $R(w_{i+1})$, if

$ErrL(u, i)$ is the minimum among all of the $ErrL$ values. Similar strategy can be used to decide the position of $R(u)$. The heuristic to distinguish neighbors and non-neighbors of u is described in Figure 4.3.

The *Neighborhood-decision* Algorithm

1. Order the right endpoints of intervals in $L\text{-part}(u)$ from left to right as $R(w_1), R(w_2), \dots, R(w_{|L\text{-part}(u)|})$ as follows :
 - (1) Let S be $L\text{-part}(u)$, T be $R\text{-part}(u)$, and i be $|L\text{-part}(u)|$.
 - (2) Let w^* be an interval in S with the largest $|N(w^*) \cap T|$, and denote $L(w^*)$ by $L(w_i)$.
 - (3) Delete w_i from S and add w_i into T .
 - (4) Decrease i by 1.
 - (5) Reiterate Step (2) to Step (4), until S is empty.
2. For $0 \leq i \leq |L\text{-part}(u)|$, let $ErrL(u, i) = |\{(u, w_j) \mid (u, w_j) \notin E(G) \text{ and } j > i\}| + |\{(w_j, w_k) \mid (w_j, w_k) \notin E(G) \text{ and } j, k > i\}| + |\{(w_i, u) \mid (w_i, u) \in E(G) \text{ and } j \leq i\}|$,
3. If $ErrL(u, t)$ is the minimum among all $ErrL$'s, we conclude that $L(u)$ should be placed between $R(w_t)$ and $R(w_{t+1})$. Let $A(u) = \{w_i \mid i > t\}$ and $LL(u) = \{w_j \mid j \leq t\}$.
4. Order the left endpoints of intervals in $R\text{-part}(u)$ from left to right as $L(v_1), L(v_2), \dots, L(v_{|R\text{-part}(u)|})$ as follows:
 - (1) Let S be $R\text{-part}(u)$, T be $L\text{-part}(u)$, and i be 1.
 - (2) Let v^* be an interval in S with the largest $|N(v^*) \cap T|$, and denote $L(v^*)$ by $L(v_i)$.
 - (3) Delete v_i from S and add v_i into T .
 - (4) Increase i by 1.
 - (5) Reiterate Step (2) to Step (4), until S is empty.
5. For $0 \leq i \leq |R\text{-part}(u)|$, let $ErrR(u, i) = |\{(u, v_j) \mid (u, v_j) \notin E(G) \text{ and } j \leq i\}| + |\{(v_j, v_k) \mid (v_j, v_k) \notin E(G) \text{ and } j, k \leq i\}| + |\{(v_i, u) \mid (v_i, u) \in E(G) \text{ and } j > i\}|$.
6. If $ErrR(u, t)$ is the minimum among all $ErrR$'s, we conclude that $R(u)$ should be placed between $L(v_t)$ and $L(v_{t+1})$, and let $B(u) = \{v_i \mid i \leq t\}$ and $RR(u) = \{v_j \mid j > t\}$.

Figure 4.3. The *Neighborhood-decision* Algorithm.

An example illustrating the idea of the neighborhood decision algorithm is shown in Figure 4.4.

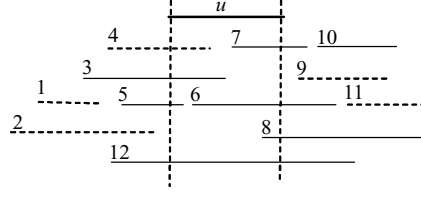


Figure 4.4. An example of neighborhood decision. In this case, the input data are noisy, but we can only depict part of errors in this figure. The solid lines and the dotted lines represent intervals overlapping u and those not overlapping u in the input data, respectively. All intervals depicted are located at the original “correct” position. Thus, 4 overlaps u originally but does not overlap u in the input data due to FN, and 10 does not overlap u originally but overlap u in the input data due to FP. Furthermore, assume that there are FPs between 1 and 11 and between 2 and 11. At the first iteration, merge the corresponding clusters of 1 and 2 into one cluster, since $OV(1,2) = 4$ is the highest. At the second and the third iterations, merge the corresponding clusters of 2 and 4, and the corresponding clusters of 9 and 11, respectively. Finally, we have that $LL(u) = \{1, 2, 4\}$ and $RR(u) = \{9, 11\}$. In the example of Figure 4.2, $FP_A(3) = 0$, $FP_B(3) = 3$, $FP_A(5) = 0$, $FP_B(5) = 2$, $FP_A(6) = 1$, $FP_B(6) = 1$, $FP_A(7) = 1$, $FP_B(7) = 0$, $FP_A(8) = 2$, $FP_B(8) = 0$, $FP_A(10) = 2$, $FP_B(10) = 0$. Thus, $A(u) = \{3, 5\}$, $B(u) = \{6, 7, 8, 10\}$. We can order right endpoints of intervals in $L\text{-part}(u)$ from left to right as $R(1)$, $R(2)$, $R(5)$, $R(4)$, $R(3)$, and order left endpoints of intervals in $R\text{-part}(u)$ from left to right as $L(7)$, $L(6)$, $L(8)$, $L(9)$, $L(10)$, $L(11)$. Furthermore, $ErrL(u,2) = 1$ is the minimum among all $ErrL$'s, and $ErrR(u,3) = 1$ is the minimum among all $ErrR$'s. Thus, we conclude that $L(u)$ should be located between $R(5)$ and $R(2)$, and $R(u)$ should be located between $L(8)$ and $L(9)$. Hence, $L\text{-part}(u)$ and $R\text{-part}(u)$ could be reclassified into $LL(u)=\{1, 2\}$, $A(u)=\{3, 4, 5\}$, $B(u)=\{9, 10, 11\}$, $RR(u)=\{6, 7, 8\}$, and the FPs and FNs relative to u itself have been corrected.

4.2 Deciding endpoint block sequence under the influence of FNs and FPs

In this section we determine the left-right block sequence within u on noisy data. The monotone collection $\{ N(w) \cap B(u) \mid w \in A(u) \}$ provides a very strong structural property for interval graphs. This property is stable enough for us to obtain a “good” left-right block sequence within interval u . In case the above collection of sets does not satisfy the monotone property, one could remove some elements and/or add some elements into the sets to make it satisfy the monotone property. We denote the removed elements as *removals* and the added elements as *fill-ins*. The removals and fill-ins could be considered as FPs and FNs, respectively. Note that it is a relative matter to decide removals and fill-ins, and there is a trade-off in determining FPs and FNs. Suppose we suspect an edge to be a FP. There are two possibilities. One is that we simply remove this edge. The other is that we let it stay, which would possibly create some FN(s) we need to fill in to preserve the monotone property. Our strategy is to detect and remove potential FPs first, and then deal with the FNs. Note that the minimum fill-in problem is NP-complete [Y1981] and a polynomial approximation for the problem has been proposed in [NS1998].

We use the FP-Screening algorithm in Figure 4.6 to determine a FP. Let $w_1, w_2, \dots, w_{|A(u)|}$ be a list in $A(u)$ ordered according to their ascending $|N(w) \cap B(u)|$ values. If $\{N(w) \cap B(u) \mid w \in A(u)\}$ is monotone, we should have $N(w_i) \cap B(u) \subseteq N(w_j) \cap B(u)$ for all $i < j$. Since data is noisy, this condition might not hold for all $i < j$, but it should hold with high probability due to low error rate. So for each $v \in N(w_i) \cap B(u)$, if $|\{j \mid j > i \text{ and } v \notin N(w_j) \cap B(u)\}| \geq 3$, the entry (w_i, v) is considered a FP. The threshold is set to be 3 since the probability that there are more than three FPs in the same interval is relatively low.

The *FP-screening* Algorithm

1. Sort intervals in $A(u)$ into a list $\{w_1, w_2, \dots, w_{|A(u)|}\}$ according to their ascending $|N(w) \cap B(u)|$ values.
2. For each $w \in A(u)$, if $|\{j \mid i < j \text{ and } v \in N(w_i) \cap B(u) \text{ and } v \notin N(w_j) \cap B(u)\}| \geq 3$, the pair of intervals (w_i, v) is considered a FP. Remove edge (w_i, v) .

Figure 4.6. The *FP-screening* algorithm

After the FPs are determined and removed, we determine fill-ins that make the collection $\{ N(w) \cap B(u) \mid w \in A(u) \}$ monotone using the following greedy strategy

shown in Figure 4.7. Initially, consider all intervals in $A(u)$ unselected. For each unselected interval w in $A(u)$, define its “fill-in cost” to be the minimum number of edges whose addition will satisfy $N(w) \cap B(u) \subseteq N(w') \cap B(u)$ for every unselected interval w' in $A(u)$, namely, define $fill-in(w) = |\{ (w',v) \mid v \in N(w) \cap B(u) \text{ and } v \notin N(w') \cap B(u) \text{ for all } w' \in A(u), w' \text{ is unselected, and } w \neq w' \}|$. Each time, select the interval, say w^* , with the minimum “fill-in cost” among unselected intervals in $A(u)$. Once w^* has been selected, add all edges counted in $fill-in(w^*)$ and mark w^* a selected interval. Reiterate the above process until all intervals in $A(u)$ are selected.

The *FN-Screening* Algorithm

1. Select an unselected interval w^* in $A(u)$ with the minimum $|fill-in(w^*)|$ value.
2. Consider each element (w',v) counted in $fill-in(w^*)$ as a FN and add edge (w',v) .
3. Mark w^* a selected interval.
4. Reiterate Step 1 to Step3 until all intervals in $A(u)$ have been selected.

Figure 4.7. The *FN-screening* Algorithm

4.3. The clustering algorithm for interval graph recognition

Finally, we summarize the algorithms of the above two sections in Figure 4.8 below. The intervals are processed according to an ascending order of their degrees. Note that an interval is considered “deleted” if it turns out that after the deletion and insertion of edges, this interval does not overlap strictly with any other interval. This is because the placement of such an interval does not affect the ordering of the other intervals. Thus, it is possible for our algorithm to create more islands (though it does not happen very often).

The *Interval-graph-clustering-test*

- 1 Neighbor Classification:
 - 1.1 Let $C(u) \leftarrow \{ w \mid N(w) \supset N(u) \}$, $D(u) \leftarrow \{ w \mid N(w) \subseteq N(u) \}$ and $STA(u) \leftarrow N(u) - C(u) - D(u)$.
 - 1.2 Construct $LL(u)$ and $RR(u)$ using the *LL-RR-classification* algorithm.
 - 1.3 Partition $STA(u)$ into $A(u)$, $B(u)$ using the *A-B-classification* algorithm.
 - 1.4 Distinguish the neighbors and non-neighbor of u using the *Neighborhood-decision* algorithm.
 - 1.5 Let u_{SL} be the special interval such that $u_{SL} \in A(u)$, and u_{SR} be the special interval such that $u_{SR} \in B(u)$.
- 2 Block sequence determination
 - 2.1 Screen out FPs using the *FP-screening* algorithm.
 - 2.2 Fill in FNs using the *FN-screening* algorithm.

- 2.3 Construct the left-right block sequence within u using the collection $\{ N(w) \cap B(u) \mid w \in A(u) \}$ and intervals in $D(u)$.
- 3 Vertex replacement:
 - 3.1 Create a new special interval u^s with $N(u^s) \leftarrow N(u_{SL}) \cup N(u) \cup N(u_{SR})$.
 - 3.2 Suppose that x is a vertex only with its right endpoint in u^s and y is a vertex only with its left endpoint in u^s . Remove edge (x, y) if it exists.
 - 3.3 Remove u, u_{SL}, u_{SR} and vertices whose left endpoints and right endpoint are both contained in u^s .

Figure 4.8. The *Interval-graph-clustering-test*.

5. Experimental Results

We conduct experiments based on synthetic data. We start with a fixed interval model and, in each experiment, randomly generate errors on the edge connections, then feed the resultant graph to our algorithm to get a left-right block ordering. Three fixed graphs of sizes 100, 200, and 400 are used. These graphs are generated randomly under the constraint that the number of endpoints an interval contains (roughly corresponds to its “coverage”) ranges from 5 to 15. The combined error rates of FPs and FNs are set to be 3%, 5% and 10%, respectively. Within each error percentage, set the ratio of the number of FPs and that of FNs to be 1 to 4, namely, every generated FP accompanies 4 FNs. For various combination of graph size and error rate, we repeat the experiment 50 times using different random seeds. The results are evaluated by comparing the resultant interval ordering from that of the original ordering, based on the measurement defined below.

Regard the position of an interval as the position of the “left endpoint” of the interval. For an interval u , let d_1 be the number of intervals ordered to the left of u but whose indices are greater than u and d_2 , the number of intervals ordered to the right of u whose indices are less than u . Let the displacement $d(u)$ of interval u be the larger of d_1 and d_2 . The displacement $d(u)$ gives an approximate measure of the distance of interval u from its “correct” position. It should be noted that defining an exact measure is difficult here since many other intervals have to be moved simultaneously in order to place a particular interval “correctly”. We use the following criterion for measuring the total deviation of the resultant ordering from the original one: If the displacement of an interval u is more than 4, we say u is a *jump interval*, which means that the position of u is quite far from its ordinary position (so the algorithm does not place u well). For example, in Figure 5.1, $d(2) = 6$ (there are 6 intervals ordered to the left of interval 2 whose indices are greater than 2), $d(6) = 1$, and $d(8) = 6$ (there are 6 intervals ordered to the right of interval 8 whose indices are less than 8). Thus, intervals 2 and 8 are jump intervals.

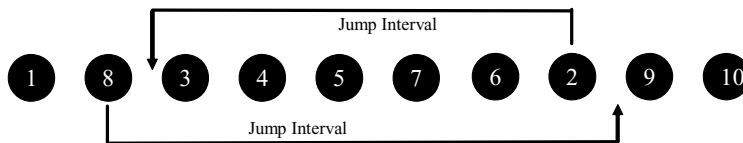


Figure 5.1. An example of jump intervals.

We measure the performance of our algorithm by counting the number of jump intervals in the resultant interval model. Table 1 lists the statistics for the number of jump intervals. As one can see, when the error rate is less than 5%, the number of jump intervals is less than 10, and even when the error rate is increased to 10%, the number of jump intervals remains less than 15 in most runs, indicating that the final interval ordering produced by the algorithm is a good approximation for the original.

Table 1. Statistics for the jump intervals.

0~5	48	34	20	38	28	4	48	40	27
6~10	1	4	17	12	17	11	1	5	13
11~15	0	0	9	0	4	23	0	3	6
16~20	1	2	1	0	1	7	1	1	3
21~25	0	0	2	0	0	1	0	0	0
26~30	0	0	1	0	0	0	0	1	0
31~35	0	0	0	0	0	1	0	0	0

An example of our experiments is shown in Figure 5.1. There are 25 intervals in the graph, and the error rate is 5%. The original interval model is shown on the left half, and the interval model generated by our program is shown on the right half. Note that some FPs and FNs are generated but not depicted in this figure.

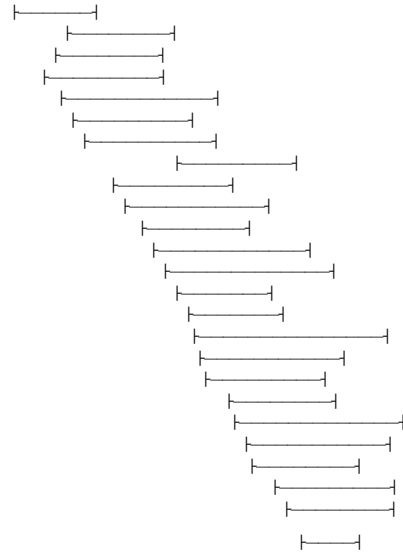
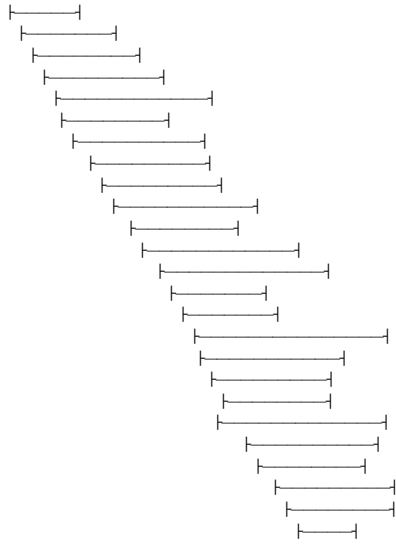


Figure 5.1 An interval graph with 25 intervals at error rate 5%

6. CONCLUDING REMARKS

The physical mapping problem in human genome research can be modeled as an interval graph recognition problem, if the overlap information is error-free. However, data collected from laboratories unavoidably contain errors. Traditional recognition algorithms can hardly be applied directly on noisy data, and related models for the imperfection are shown to be NP-hard. In this paper we propose a clustering algorithm for interval graph test on noisy data. The design of our algorithm is based on the philosophy of local structure matching. For two typical error types FPs and FNs, we check the neighborhood data to see whether they conform “approximately” to a particular local structure dictated by interval graphs to determine whether overlapping information is valid or noisy. The experimental results show that, when the error percentage is small, our clustering algorithm is robust enough to discover certain errors and to correct them automatically most of the time.

References

- [BF1996] H. L. Bodlaender, B. de Fluiter. On Intervalizing k-Colored Graphs for DNA Physical Mapping. *Discrete Applied Math.s*, 71, 55-77, 1996.
- [BL1976] K.S. Booth and G.S. Lueker, Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-tree Algorithms, *J. Comput Syst. Sci.* 13 , 335-379, 1976.
- [CB1989] A.V. Carrano, P.J. de John, E. Branscomb, T. Slezak, B.W. Watkins. Constructing chromosome and region-specific cosmid map of the human genome, *Genome* 31, 1059-1065, 1989.
- [CS1989] A. Coulson, J. Sulston, S. Brenner, J. Karn. Toward a physical map of the genome of the nematode, *Caenorhabditis Elegans. Proc. Natl. Acad. Sci. USA* 83, 7821-7825, 1987.

- [FHW1993] M.R. Fellows, M.T. Hallett, H.T. Wareham. DNA Physical Mapping: Three Ways Difficult. ESA 1993 (LNCS 726)
- [GC1987] R.M. Gemmil, J.F. Coyle-Morris, F.D. Jr. McPeck, L.F. Wara-Urbe, F. Hecht. Construction of long-range restriction maps in human DNA using pulsed field gel electrophoresis. *Gene Anal. Technol.* 4, 119-131, 1987.
- [GGKS1995] P.W. Goldberg, M.C. Golumbic, H. Kaplan and R. Shamir. Four strikes against physical mapping of DNA. *J. Comput. Biol.* 2, 139-152.1995.
- [GK1994] M.C. Golumbic, H. Kaplan and R. Shamir, On the Complexity of DNA Physical Mapping, *Advances in Applied Mathematics* 15, 251-261, 1994.
- [GKS1994] M.C. Golumbic, H. Kaplan, R. Shamir. On the complexity of DNA physical mapping. *Advances in Applied Mathematics* 15, 251—261, 1994.
- [GO1990] E.D. Green and M.V. Olson. Chromosomal region of the cystic fibrosis gene in yeast artificial chromosomes: a model for human genome mapping, *Science* 250, 94-98, 1990.
- [Hsu 1992] W.L. Hsu, A simple test for interval graphs, *LNCS* 657, 11-16, 1992.
- [HM 1991] W.L. Hsu and T. H. Ma, Substitution Decomposition on Chordal Graphs and Applications, *LNCS* 557, 52-60, 1991.
- [KM89] N. Korte and R. H. Möhring, *An Incremental Linear-Time Algorithm for Recognizing Interval Graphs*, *SIAM J. Comput.* 18, 1989, 68-81.
- [KS1996] H. Kaplan, R. Shamir. Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques. *SIAM J. Comput.* 25:3, 540-561,1996.
- [KS1999] H. Kaplan, R. Shamir. Bounded Degree Interval Sandwich Problems. *Algorithmica* 24:96-104, 1999.
- [KST1999] H. Kaplan, R. Shamir, R.E. Tarjan. Tractability of Parameterized Completion Problems on Chordal, Strongly Chordal, and Proper Interval Graphs. *SIAM Journal on Computing*, 28(5), 1906-1922, 1999.

- [LH 2003] W.F. Lu and W.L. Hsu. A test for the Consecutive Ones Property on Noisy Data. To appear in *Journal of Computational Biology*.
- [MCZ1987] F. Michiels, A.G. Craig, G. Zehetner, G.P. Smith, and H. Lehrach. Molecular approaches to genome analysis: A strategy for the construction of ordered overlapping clone libraries. *Comput. App. Biosci.* 3(3),203-210.1987.
- [NS1998] A. Natanzon, R. Shamir, R. Sharan. A Polynomial Approximation Algorithm for the Minimum Fill-In Problem. *STOC* 1998, 41-47.
- [OD1989] M.V. Olson, E. Dutchik, M.Y. Graham, G.M. Brodeur, C. Helms, M. Frank, M. MacCollin, R. Acheinman, T. Frand. Random-clone strategy for genomic restriction mapping in yeast, *Proc. Natl. Acad. Sci. USA* 83, 7826-7830, 1989.
- [OHCD1985] M.V. Olson, L. Hood, C. Cantor, and D. Botstein, A common language for physical mapping of the human genome, *Science* 234,1434-1435, 1985.
- [S 1997] Stoer, Mechthild and Wagner, Frank. A simple min-cut algorithm. *Journal of the ACM*, v.44 n.4, p.585-591, 1997.
- [Y1981] Yannakakis, M. Computing the Minimum Fill-In is NP-Complete, *SIAM J. Alg. Disc. Meth* 2, 1981