A Test for the Consecutive Ones Property on Noisy Data

Wei-Fu Lu

Institute of Computer and Information Science

National Chiao Tung University, Hsin-chu, Taiwan, ROC

email: gis84812@cis.nctu.edu.tw

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

Keyword: consecutive ones property, physical mapping, DNA sequence assembly, probe hybridization

* The corresponding author

Abstract

A (0,1)-matrix satisfies the *consecutive ones property* (*COP*) for the rows if there exists a column permutation such that the ones in each row of the resulting matrix are consecutive. The consecutive ones test is useful for DNA sequence assembly, for example, in the *STS* content mapping of YAC library, and in the Bactig assembly based on *STS* as well as *EST* markers. 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 for this problem, which has the following advantages:

1. If the original matrix satisfies the COP, then the algorithm will produce a column ordering realizing it without any fill-in.
2. Under moderate assumptions, the algorithm can accommodate the following four types of errors: FNs, FPs and NPs and CCs. Note that in some cases (low quality EST marker identification), NPs occur because of repeat sequences.
3. In case some local data is too noisy, our algorithm could likely discover that and suggest additional lab work that could reduce the degree of ambiguity in that part.
4. A unique feature of our algorithm is that, rather than forcing all probes to be included and ordered in the final arrangement, our algorithm would delete some probes. Thus, it could produce more than one contig. The gaps are created mostly by noisy columns.

In summary, we have modified previous rigid algorithms for testing consecutive ones property into one that can accommodate clustering techniques, and produces satisfactory approximate probe orderings for most data.

1. Introduction

A (0,1)-matrix satisfies the *consecutive ones property* (*COP*) for the rows if there exists a column permutation such that the ones in each row of the resulting matrix are consecutive. Booth and Lueker (1976) invented a data structure called *PQ*-trees to test the *COP* of (0,1)-matrices in linear time. However, the implementation of *PQ*-tree algorithm is quite complicated and it is unclear how one can modify this rigid algorithm to accommodate errors in the input data (Alizadeh et al. (1994 and 1995)). To avoid the use of PQ-trees, Hsu (2002) designed a simple off-line test for the *COP*, which does not use *PQ*-trees. The test in Hsu (2002) requires the computation of a global decomposition tree and a special row ordering before the actual consecutive ones test (*COT*) can take place and therefore, is not suitable for noisy data either. However, some idea in Hsu (2002) is quite robust and will be modified in this paper to deal with input data that contains errors.

1.1 Applications of the COT

An important application of the *COT* is in the construction of physical maps for human DNA sequences. The term "physical mapping" has come to mean the determination of the order between landmarks in stretches of DNA by physicochemical and biochemical methods. Several types of landmarks have been used in physical maps, such as restriction fragment length polymorphism (RFLP), restriction enzyme sites, sequence tagged sites (*STS*s), expressed sequence tags (ESTs), and single nucleotide polymorphisms (SNPs). The construction of physical maps is generally accomplished as follows. Long DNA sequences are separated into smaller fragments (called *clones*). A number of landmarks (*probes* or markers) are tested for their presence or absence in the clones. Given the collection of probes each clone has been attached to, one tries to order the probes in such a way that probes belonging to the same clone are consecutive. These will give us the relative positions of the clones in the DNA sequence. The error free version of the mapping problem can be viewed as the consecutive ones testing problem. However, the problem becomes much harder when the data contain errors.

The present paper focuses on the Sequence Tagged Site (*STS*) and Expressed Sequence Tag (ESTs) mapping strategies, which are widely used for physical mapping within the Human Genome Project (Palazzolo et al. 1991 and Mizukami et al. 1993). Recall that the *STS*s appear uniquely in the genome and the *EST*s may be non-unique. In current sequence assembly effort, High Throughput Genomic (HTG) Sequences division was created to accommodate a growing need to make 'unfinished' genomic sequence data rapidly available to the scientific community. At the current density of markers and the quality of the human genome draft, a mapping program may not have sufficient information to get a complete physical map. Usually there are multiple islands rather than a single contig. Lu et al. (2001) have taken advantage of the expressed sequence tag (EST) matches as markers to increase marker density. Some proper low quality (such as 70%) matches will be used to fortify the map assembly. The incorporation of *EST* matches will not only order BAC clones, but also the fragments within a clone. Moreover, the relation of two non-overlapping clones can be determined if they share *EST*s that belong to the same UniGene cluster. We shall discuss this further in Section 5.

As a side interest, another application of the COT is on the storage problem of sparse (0,1)-matrix. If a given (0,1)-matrix satisfies the COP, then after a suitable column permutation, the ones in each row form a consecutive block. Thus, one can record the ones in each row by taking down the start and end entry of its block of ones. Given a matrix that is slightly out-of-kilter, how does one modify its zeros and ones to satisfy the COP?

1.2 Previous Approaches for Dealing with Errors

There are four possible types of errors in hybridization data for physical mapping: false negatives, false positives, non-unique probes and chimeric clones. A *false negative* (FN) is an entry of 0 that should actually be 1. A *false positive* (FP) is an entry of 1 that should actually be 0. A *non-unique probe* (NP) is a probe sequence that occurs more than once in the DNA sequence (in the original clone-probe incidence matrix it would combine several columns into a false column). Two (or more) clones that stick together at their ends form a *chimeric clone* (CC) (it would combine several rows of the original clone-probe incidence matrix into a false row). Experimental errors could create FPs, FNs and CCs; repeat sequences (or chromosomal duplications, Eichler (2002)) would likely create “NPs.” These errors need to be detected and corrected in order to yield the original valid clone-probe matrix. To tackle these problems would require a different philosophy in designing algorithms. In fact, Karp (1993) posted this as a major challenge for computer science. Several related problems have been proved to be NP-hard (Golumbic et al. 1994, Golumbic and Shamir 1993 and Yannakakis 1981).

There are many related researches done on this problem. Alizadeh et al. (1994) suggested maximum-likelihood functions modeling the physical mapping problem, and solved this problem based on the local search. Another method is to approximate the maximum-likelihood function by a well-studied combinatorial problem such as the Hamming-Distance Traveling Salesman Problem (Alizadeh et al. 1994, 1995, and Greenberg and Istrail 1995). Christof et al. (1997) formulated the maximum-likelihood model as a weighted betweenness problem using branch-and-cut algorithms to bear on physical mapping. Jain and Myers (1997) converted the physical mapping problem into 0/1 linear programming (LP) problem. Mayraz and Shamir (1999) constructed physical maps using greedy method based on Bayesian overlap score. Methods for filtering the data has also been offered as an attempt to remove typical errors such as FPs and CCs (Gillett et al. 1995 and Mott et al. 1994).

However, none of the approaches in the literature can deal with all four types of errors simultaneously. Almost all of them assumed there is no NP.

1.3 The Nature of Error Treatment

Suppose the error percentage is 5%. The challenge is then to discover the 95% correct information versus the 5% incorrect information automatically. There are two difficulties we must face:

1. Different types of errors could be intertwined together as mentioned above. Sometimes it is possible to transform certain recognition problems into optimization problems by defining some "distance measure". For example, if we restrict the error types to be FPs and FNs only, one can certainly propose an obvious distance measure for the *COP* by asking "what is the least number of (0,1)-flips required for the given matrix to satisfy the *COP*." But such an approach usually suffers from the following two unpleasant phenomena: (1) The problem of finding this least number would likely become *NP*-hard; (2) Even if one can find the best flips, the data in the resultant matrix might not make much biological sense.

When all four types of errors occur simultaneously, the existence of FPs and FNs makes it even more ambiguous to detect CCs and NPs. The dilemma is that, if one could not identify CCs and NPs correctly, it would be difficult to identify FPs and FNs, and the whole ordering could be corrupted. This error-mixed problem has multiple objectives since we want to (1) minimize (in fact, eliminate) the number of CCs; (2) minimize the number of NPs; and simultaneously (3) minimize the number of (0,1)- flips for the given matrix to satisfy the *COP*, etc. Thus, it would be difficult just to define an associated "single objective optimization problem" for approximation. Even if one could formulate such an optimization version, it would most likely be NP-hard and the approximate solutions to such optimization problems might not make much biological sense.

2. The errors might not be uniformly distributed. In other words, some local data might be a lot noisier than we expected on the average. An improper treatment of such local noise (such as a NP or a CC) could corrupt the whole probe arrangement. Therefore, any global approach for the COT ignoring local variations could be catastrophic. Most of the previous approaches would produce a "complete" arrangement of all the probes. However, in our approach, if it appears that a probe creates a disrupting behavior for its neighbors, it could be deleted from further consideration. Thus, we shall delete a small percentage of the probes; produce a few contigs (which we have more confidence in) rather than one contig with a complete permutation of the probes as in most other methods. Such a measure is installed to prevent possible disastrous result and is regarded as a key feature of our algorithm.

1.4 Our Approach

In view of the difficult nature of error treatment, we opt to maintain a stable local structure through clustering techniques in our algorithm. The main idea of our algorithm is based on the column contraction in Hsu (2002). We do not set any "global" objective to optimize. Rather, our algorithm tries to maintain the local monotone structure, namely, to minimize the deviation from the local monotone property as much as possible. 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.

We assume the error rate is reasonably small, say less than 10%; the clone coverage is large enough, and most clones contain enough number of probes in order for the feature structure to be more prominent. In case the latter assumptions are too strong, we shall modify some threshold values in the algorithm accordingly. Our philosophy is that, in order to determine whether a piece of information (such as two clones overlap in some probe) is a valid signal or a noise we check the neighborhood data to see whether they conform "approximately" to a particular local structure dictated by the problem. The probability that an isolated piece of spurious information will have 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 (0,1)-pattern in the neighborhood will emerge which will allow us to weed out most errors. If some crucial piece of information is missing or some local data is too noisy, our *COT* can often detect this and suggest additional lab work that could reduce the degree of ambiguity for that part.

Our clustering algorithm has the following features:

1. If the original matrix satisfies the COP, then the algorithm will produce a column ordering realizing it without any fill-in.
2. Under moderate assumptions, the algorithm can accommodate the following four types of errors: FNs, FPs and NPs and CCs. Note that in some cases (low quality EST marker identification), NPs occur because of repeat sequences.
3. In case some local data is too noisy, our algorithm could likely discover that and suggest additional lab work that could reduce the degree of ambiguity in that part.
4. A unique feature of our algorithm is that, rather than forcing all probes to be included and ordered in the final arrangement, our algorithm would delete some probes. Thus, it could produce more than one contig. The gaps are created mostly by noisy columns.

Experimental results (to be described in Section 5) 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.

In summary, we have modified previous rigid algorithms for testing consecutive ones into one that can accommodate clustering techniques, and produces satisfactory approximate probe orderings for most data. The remaining sections are arranged as follows. Section 2 gives the basic definitions. A *COT* modified from Hsu (2002) is discussed in Section 3, which forms the basis for our error-tolerant algorithm. Section 4, the main part of this paper, illustrates how we deal with these four types of errors in the input data. Section 5 gives the experiemntal results.

2. Basic Definitions

Let M be an $m \times n$ (0,1)-matrix whose total number of ones is r. Denote by $R(M)$ the set of rows of M and $C(M)$ the set of columns. Then $|R(M)| = m$ and $|C(M)| = n$. We use v to denote a general column and u to denote a general row. For each row u in $R(M)$, let $CL(u)$ be the set of columns that contain a nonzero in this row. For any subset R' of $R(M)$ define $CL(R')$ to be the set of columns that have a nonzero entry in a row of R'. For each column v in $C(M)$, let $RW(v)$ be the set of rows that contain a nonzero entry in this column. For any subset C' of $C(M)$, define $RW(C')$ to be the set of rows that contain a nonzero entry in a column of C'. A subset of $CL(u)$ is denoted by a *sub-row* of row u. Denote the *size* of a row u by $|CL(u)|$, and the *size* of a column v by $|RW(v)|$. Label the rows according to an ascending order of their sizes, and the columns in arbitrary order. For convenience, we shall not distinguish between the two terms, "row" (resp. "column") and its corresponding "row index" (resp. "column index").

Two rows x and y are said to *overlap* if they have a nonzero entry in a common column, and they are said to be *independent* if they do not share a nonzero entry. A row x is said to *be contained in* another row y if no entry of x is greater than the corresponding entry of y, and a containment is said to be *proper* if at least one entry of x is less than the corresponding entry of y. Two rows x and y are said to *overlap strictly* if they overlap but neither is contained in the other. The strictly overlapping relationships on the rows play an important role in checking the consecutive ones property.

3. A modified Consecutive Ones Test

Although there exist several linear time algorithms for the *COT*, there is no obvious way to modify these algorithms to incorporate small errors in the data. For example, in Booth and Lueker's *PQ*-tree algorithm, a single error would terminate the construction of the *PQ*-tree. A similar phenomenon occurs in Hsu's decomposition approach.

There are two main ideas in Hsu's (2002) approach: (1) consider a local monotone triangular property as described in this paper; (2) find a "good" row ordering to be tested iteratively rather than using the breadth-first-order as in

Fulkerson and Gross (1965). The modified version in Section 2 does not require a good row ordering and relaxes the running time to O(n^2), but maintains the implementation of the monotone triangular property. The increase in time complexity allows us to restore the triangular property under the influence of false positives and false negatives. We shall first describe a quadratic time *COT* based on Hsu (2002) in this section. The main reason to describe such a test is to define the “monotone” neighborhood structure, which is a key feature for our clustering algorithm.

The main idea of this *COT* can be described as follows. The rows are processed according to an ascending order of their sizes (rather than following a specific pre-computed order as in Hsu (2002)). During each iteration, we determine the unique order of all columns within $CL(u)$. Since smaller rows are processed before larger ones, we can guarantee that whenever a row u is being processed, any row that is properly contained within u originally must have been processed, and the unique ordering of columns in $CL(u)$ can be obtained. If $|CL(u)| > 1$, then at the end of the iteration, all but two columns are deleted and a special row containing these two undeleted columns is generated. Thus, the matrix is further reduced. The reason for creating the special row is to preserve the *COP* for the original matrix. This process continues until all original rows are processed. The main iteration of the algorithm is described in Figure 1.

At each iteration, the columns of $CL(u)$ are partitioned into sets, say S_1, S_2, ..., S_d, with a fixed left-right ordering (however, the column orders within each set are arbitrary). Such a partition naturally *induces* a collection of column orderings in which columns in each S_i are arranged consecutively in an arbitrary ordering and the column groups of the S_is are arranged from left to right according to the order S_1, S_2, ..., S_d. To describe the algorithm we need the following definitions.

Definition 3.1. *A collection of sets is said to be monotone if for every two sets S_1, S_2 in the collection, either $S_1 \supseteq S_2$ or $S_2 \supseteq S_1$.*

Lemma 3.2. *If a collection of sets S_1, S_2, …, S_n is monotone (and non-increasing in size), then the collection of sets $\{ T(x) \mid x \in S_1 \cup S_2 \cup \ldots \cup S_n\}$ is monotone, where $T(x)=\{ i \mid x \in S_i\}$.*

Definition 3.3. A row u is said to be *compatible* with a column partition S_1, S_2, ..., S_d in which the S_js are ordered from left to right if it satisfies that $CL(u) \cap [S_1 \cup S_2 \cup \ldots \cup S_d] \neq \emptyset$ and

1. Either (1) $CL(u) \subseteq [S_1 \cup S_2 \cup ... \cup S_d]$, in which case let S_{j1} (resp. S_{j2}) be the leftmost (resp. rightmost) set having nonempty intersection with $CL(u)$. Then all sets in between (but excluding) S_{j1} and S_{j2} are contained in $CL(u)$.

Or (2) $CL(u) - [S_1 \cup S_2 \cup ... \cup S_d] \neq \emptyset$, in which case let S_j be any set having nonempty intersection with $CL(u)$. Then either all sets to the right of S_j are contained in $CL(u)$ or all sets to the left of S_j are contained in $CL(u)$.

The main iteration of our algorithm is described in Figure 1 below:

```
The COT Algorithm: Processing an original row $u$
1. If $|CL(u)| \leq 1$, delete $u$. Proceed to the next row.
2. Mark all columns in $CL(u)$. Determine $RW(v)$ for each $v$ in $CL(u)$. For each row $w$ in some $RW(v)$
      for $v$ in $CL(u)$ ($w$ is a row that overlaps $u$), count $|CL(w) \cap CL(u)|$.
3. Based on $|CL(w) \cap CL(u)|$, construct the following set: $C(u) = \{ w \mid w$ properly contains $u \}$, $D(u) =$
      $\{ w \mid CL(w) \neq \emptyset$ and $w$ is properly contained in $u \}$, $I(u) = \{ w \mid w$ is identical to $u \}$ and $STA(u) =$
      $\{w \mid w$ overlaps strictly with $u\}$.
4. Let $u^*$ be a row in $STA(u)$ with the largest $|CL(u^*) \cap CL(u)|$.
      Let $v^*$ be a column in $CL(u)-CL(u^*)$ with the largest $| STA(u) \cap RW(v) |$.
5. Let $A(u) = STA(u) \cap RW(v^*)$.
      Let $B(u) = STA(u) - A(u)$.
6. Let $v^A$ be a column in $CL(A(u)) \cap CL(u)$ with the largest $|RW(v^A) \cap A(u)|$.
      Let $v^B$ be a column in $CL(B(u)) \cap CL(u)$ with the largest $|RW(v^B) \cap B(u)|$.
7. Partition $CL(u)$ using sub-rows in the three sets $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$, $\{ CL(w) \cap CL(u) \mid$
      $w \in B(u) \}$ and $\{ CL(w) \cap CL(u) \mid w \in D(u) \}$ to obtain a unique partition. We need to check the
      following:
      7.1 The two sets of sub-rows $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$, $\{ CL(w) \cap CL(u) \mid w \in B(u) \}$
            are monotone and $CL(u)$ can be uniquely partitioned with $v^A$, $v^B$ placed at one end,
            respectively. Every sub-row in one set is compatible with the column partition determined
            by the other set.
      7.2 Every row in $D(u)$ is compatible with the column partition determined by the above two
            sets and the remaining rows in $D(u)$.
      7.3 If there is any violation, then $M$ does not satisfy the $COP$ and we can terminate the
            algorithm.
8. Delete all columns in $CL(u)$ except $v^A$ and $v^B$. Construct a new specials row $u^S$ with
      $CL(u^S)=\{v^A,v^B\}$. Proceed to the next row.
```

Figure1. The *COT* Algorithm.

We shall prove in Theorem 3.7 that, for a matrix *M*, the *COT* algorithm decides whether *M* satisfies the *COP* correctly. Several lemmas are needed for the proof of Theorem 3.7. Below, we shall follow the notations used in the *COT* algorithm.

Lemma 3.4. *If a (0,1)-matrix M satisfies the COP, then the collections* $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$*and* $\{ CL(w) \cap CL(u) \mid w \in B(u) \}$ *are monotone.*

Proof. We shall prove that $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$ is monotone. The proof that $\{ CL(w) \cap CL(u) \mid w \in B(u) \}$ is monotone is symmetric. Let π be a column permutation of *M* that realizes the *COP*. Since columns in $CL(u)$ are consecutive in π, columns in $CL - CL(u)$ could be partitioned into two disjoint sets, say $LF(u)$ and $RT(u)$, such that the π-index of each column in $LF(u)$ is less than the π-index of each column in $CL(u)$, and the π-index of each column in $RT(u)$ is greater than the π-index of each column in $CL(u)$. Without loss of generality, assume $CL(u^*) - CL(u) \subseteq RT(u)$, where u^* is a row in $STA(u)$ with the largest $|CL(u^*) \cap CL(u)|$. It is easy to check that $A(u) = \{ w \mid CL(w) - CL(u) \subseteq LF(u), |CL(w) \cap CL(u)| < |CL(u)| \}$ and $B(u) = \{ w \mid CL(w) - CL(u) \subseteq RT(u), |CL(w) \cap CL(u)| < |CL(u)| \}$. Figure 2 gives an example of the sets $A(u)$ and $B(u)$. Since *M* satisfies the *COP*, the π-index of columns in $CL(w) \cap CL(u)$ for $w \in A(u)$ are $\pi^{-1}(s), \pi^{-1}(s+1), \ldots, \pi^{-1}(s + |CL(w) \cap CL(u)| -1)$. For any two rows w_1 and w_2 in $A(u)$, if $|CL(w_1) \cap CL(u)| \leq |CL(w_2) \cap CL(u)|$, then $CL(w_1) \cap CL(u) \subseteq CL(w_2) \cap CL(u)$. Hence, $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$ is monotone. ▮

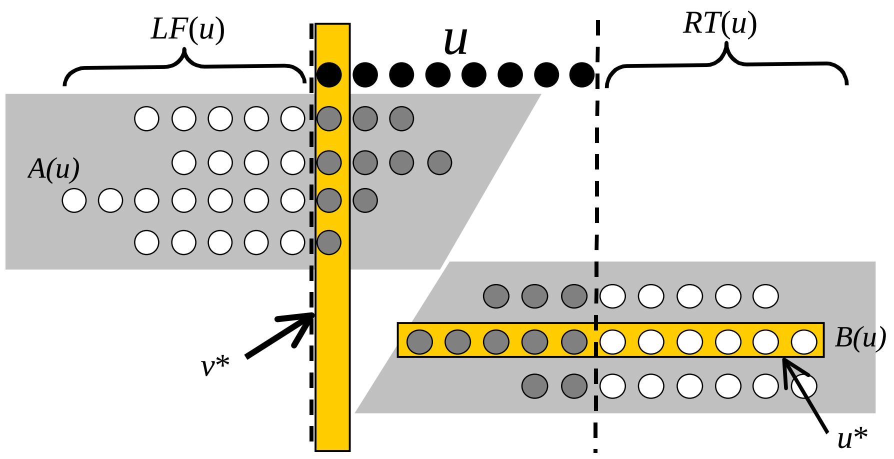

Figure 2. The sets *A*(*u*) and *B*(*u*) of a matrix satisfying the COP

Since $STA(u)$ is the disjoint union of $A(u)$ and $B(u)$, we have

Corollary 3.5. *If M satisfies the COP, then* $\{ CL(w) \cap CL(u) \mid w \in STA(u) \}$ *can be partitioned into two monotone collections uniquely.*

Lemma 3.6. *Let $M_{A(u)}$ be the submatrix of M consisting of rows in $A(u)$ and columns in $CL(A(u)) \cap CL(u)$. If $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$ is monotone, then columns in $CL(A(u)) \cap CL(u)$ can be partitioned uniquely with v^A at one end, such that the column partition induces a collection of column orderings that realize the COP of $M_{A(u)}$. Similarly, columns in $CL(B(u)) \cap CL(u)$ can be partitioned uniquely with v^B at one end of $M_{B(u)}$.*

Proof. If $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$ is monotone, then $\{ A(u) \cap RW(v) \mid v \in CL(A(u)) \cap CL(u)\}$ is monotone from lemma 3.2. Partition columns in $CL(A(u)) \cap CL(u)$ into ordered set $S_1, S_2, \ldots, S_n$ such that for any two columns v_1 and v_2 in $CL(A(u)) \cap CL(u)$, we have the following:

1. For v_1 and v_2 in the same set, $A(u) \cap RW(v_1) = A(u) \cap RW(v_2)$

2. For $v_1 \in S_i$ and $v_2 \in S_j$ such that $i < j$, $A(u) \cap RW(v_1) \subset A(u) \cap RW(v_2)$.

Thus, the above partition $S_1, S_2, \ldots, S_n$ is an column partition with v^A at one end, which induces a collection of column orderings realizing the *COP* of $M_{A(u)}$. The proof for $M_{B(u)}$ is symmetric. ■

Theorem 3.7. *A (0,1)-matrix M satisfies the COP iff the following conditions hold at each iteration of the COT algorithm :*

The two sets of sub-rows $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$, $\{ CL(w) \cap CL(u) \mid w \in B(u) \}$ are monotone and can be uniquely partitioned with v^A, v^B placed at one end, respectively. Every sub-row in one set is compatible with the column partition determined by the other set.

Every row in $D(u)$ is compatible with the column partition determined by the above two sets and the remaining rows in $D(u)$.

Proof. If a matrix M satisfies the *COP*, from lemma 3.4 one can easily check the necessity of these conditions. Note that condition 2 simply indicates that the order of rows in $D(u)$ considered for the partition is immaterial.

Conversely, we shall use induction on $|CL(M)|$. Assume the statement is true for matrices smaller than M. From lemma 3.6, if these conditions are satisfied at every iteration, then for each row u processed, The *COT Algorithm* determines a partition, say $P(u)$, of $CL(u)$ as follows. Form a unique partition of $CL(A(u)) \cap CL(u)$ based on the monotone collection $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$ in which v^{A} (in step (7) of the *COT Algorithm*) is at one end (this forces a unique partition). Likewise, form a unique partition of $CL(B(u)) \cap CL(u)$ with v^B at one end. Refine these two partitions based on their overlapping compatible rows. Now, further refine this partition by bringing in compatible rows in $D(u)$ one by one.

Let k be the first iteration that some columns are deleted in step (8) of *Algorithm COT*. Let M' be the reduced matrix. Since $|CL(M')| < |CL(M)|$ and the two conditions in Theorem 3.7 are satisfied at every iteration after iteration k, by the induction hypothesis, M' satisfies the *COP*. Consider any column permutation of M' realizing the *COP* for the rows, say v'_1, v'_2, … , v'_d. Denote the partition $P(u_k)$ by S'_1, S'_2, … ,S'_h, where the two special columns $v_k^A \in S'_1$and $v_k^B \in S'_h$. Since $\{v_k^A, v_k^B\}$ is the column set of the special row u^S, we must have $v_k^A = v'_j$ and $v_k^B = v'_{j+1}$ for some j. Now, insert the deleted columns of $CL(u_k)$ via $P(u_k)$ back to get a column partition v'_1, v'_2, …v'_{j-1}, S'_1, S'_2, … ,S'_h, v'_{j+1}, … , v'_d that realizes the *COP* for *M*. Therefore, these conditions are also sufficient. █

If the given matrix *M* satisfies the *COP*, then the *COT Algorithm* will yield a column permutation with consecutive ones in each row. Otherwise, the algorithm will terminate in step (7) at some iteration.

4. Treating the Errors

In this section, we present an error-tolerant version of the *COT* algorithm. We shall simultaneously consider the following four types of errors in the data set: NPs, CCs, FPs and FNs. We 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.

Note that the position of a FP in a clone may be far from those of the other probes in the clone. This kind of FPs will be denoted by *remote FPs*. Figure 3 gives an example of remote FPs. Since remote FPs, NPs and CCs are most disruptive, we shall try to eliminate them first. After they have been removed, we then remove the remaining FPs and FNs. In these clustering algorithms, we need to set different threshold values for detecting various errors. Whenever possible, we shall provide motivations for these threshold values by proving some lemmas for the ideal situations. Since we do not generate a lot of long clone segments in the simulated data, decomposition is usually unnecessary.

Because different types of errors could tangle with each other, the order of the error elimination process is very crucial. This is summarized in the *Error-Tolerant COT* algorithm in Section 4.5. It is interesting to note that, in our approach, the algorithm for eliminating NPs and CCs simultaneously discover remote FPs. Hence, it is an integral part of the is alongside the elimination of. Namely, regardless of whether there are NPs or CCs in the data, we need to execute the algorithms in Sections 4.1 and 4.2 just to ensure remote FPs are eliminated.

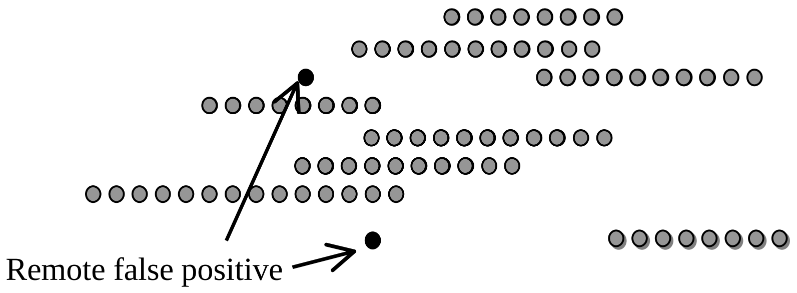

Figure 3. An example of remote false positives

4.1. Screening out Non-unique probes and Related Remote False Positives

Consider a NP *v*. Roughly speaking, because probe *v* appears in different places of the DNA sequence, the rows containing *v* will form two or more clusters based on their neighbor overlapping relationships. Therefore, for each column *v'*, we adopt a clustering method, called row-neighbor clustering, to determine if there exist two or more clusters for the rows containing column *v'*, in order to decide whether *v'* is a NP. Figure 4 gives an example of two clusters NUP_1 and NUP_2 containing the NP *v*.

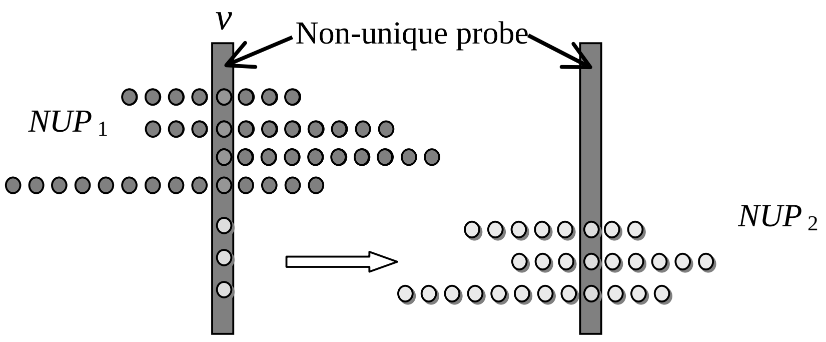

Figure 4. An Example of a non-unique probe.

Sometimes, a remote FP may generate another cluster, for example, the cluster NUP_2 in Figure 5. Those can be screened out using the row-neighbor clustering analysis. Note that some local (close-by) FPs may not generate another clusters, and could not be detected using the row-neighbor clustering analysis. We shall detect those using the monotone structure described in section 4.4.

From the above discussion, we shall screen out NPs and related remote FPs using a clustering analysis on $RW(v)$.

Definition 4.1.1. *The overlap function $OV(u,w)$ between two rows u and w in $RW(v)$ is the number of columns that are contained in both u and w, namely, $|CL(u) \cap CL(w)|$. If $OV(w_1,u) > OV(w_2,u)$, we say w_1 is closer to u than w_2. The overlap function $OV(u,S)$ between row u and a set S of rows is defined to be $\sum_{\forall v \in S} OV(u,v) \big/ |S|$, which measures the degree of overlapping between row u and rows in S.*

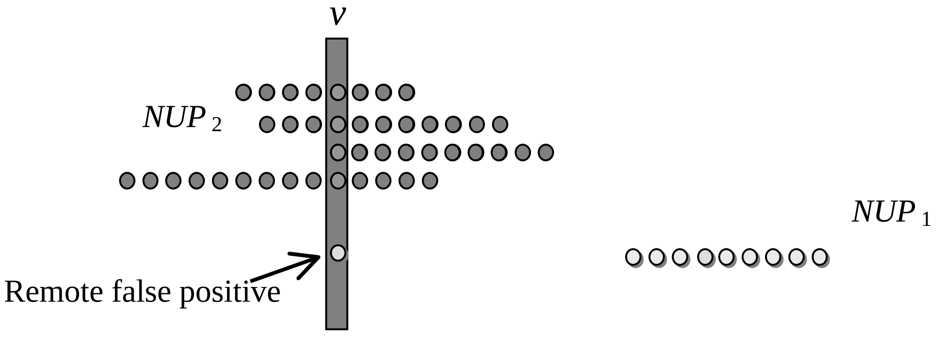

Figure 5. An example of a cluster generated by remote FPs

Lemma 4.1.2. *Let M be a (0,1)-matrix satisfying the COP in which each row has size L. Let π be a column permutation of M that realizes the COP. For any two columns v_1 and v_2 of M, if $|\pi^{-1}(v_1) - \pi^{-1}(v_2)| \geq 2L - 2$, then $OV(u_1,C) > OV(u_2,C)$ for any two rows u_1 in $RW(v_1)$, u_2 in $RW(v_2)$, and any subset C of $RW(v_1)$.*

Proof. Since $u_1 \in RW(v_1)$, for all $w \in RW(v_1)$, we have $v_1 \in CL(u_1) \cap CL(w)$ and $OV(u_1,RW(v_1)) > 1$. Without lost of generality, assume $\pi^{-1}(v_1) < \pi^{-1}(v_1)$, then $\pi^{-1}(v_2) \geq 2L - 2 + \pi^{-1}(v_1)$ and for any $u_2 \in RW(v_2)$ and $w \in RW(v_1)$, $|CL(u_2) \cap CL(w)| \leq 1$. Thus, we have $OV(u_2,RW(v_1) \leq 1 < OV(u_1,RW(v_1)$ and the lemma is proved. ▌

Lemma 4.1.3. *Let M be a (0,1)-matrix satisfying the COP in which each row has size L. Let π be a column permutation of M that realizes the COP. Let v_1 and v_2 be any two columns of M such that $|\pi^{-1}(v_1) - \pi^{-1}(v_2)| \geq 2L - 2$. Let C be a subset of $RW(v_1)$ and u a row not in C with maximum $OV(u,C)$. If $u \in RW(v_2)$, then $C = RW(v_1)$.*

Proof. Assume that $u \in RW(v_2)$, and $C \neq RW(v_1)$.islet w be a row *in* $RW(v_1) - C$. Since u is a row not in C with maximum $OV(u,C)$, we have $OV(u,C) > OV(w,C)$. However, since $u \in RW(v_2)$ and $w \in RW(v_1)$, this is contradictory to lemma 4.1.2. ▌

According to the above property, we can easily derive the following for a NP

Corollary 4.1.4. *Assume there are no CCs, FPs and FNs, and there is a NP appearing in positions p_1 and p_2 of the DNA sequence. Let NUP_1 be the cluster of rows that contain the position p_1 and NUP_2 be the cluster of rows that contain the position p_2. Let NUP' be a subset of NUP_1 and u a row not in NUP' with maximum OV(u,NUP'). If $u \in NUP_2$, then NUP' = NUP_1.*

Our clustering algorithm constructs clusters one by one as follows. First, select the shortest unselected row to be the initial element of a cluster. Then consider the row, say u_{nearest}, that has the maximum overlap value with the cluster *NUP*. If $OV(u_{\text{nearest}}, NUP)$ is greater than a threshold value d, add u_{nearest} into the cluster. Otherwise, terminate the construction for the current cluster and start a new cluster. In our experiments, d is set to be 2, which seems to get good clustering performance. The details are described in Figure 6.

```
The ROW-NEIGHBOR CLUSTERING Algorithm.
 1. Let i be1 and $S = RW(v)$. Set the threshold value to be 2.
 2. Let $NUP_i = \{ u_{init} \}$, where $u_{init}$ is the shortest row in S. $S \leftarrow S - \{ u_{init} \}$.
 3. Let $u_{nearest}$ be the row in S with maximum $OV(u_{nearest}, NUP_i)$. If $OV(u_{nearest}, NUP_i) > d$, then $NUP_i \leftarrow$
    $NUP_i \cup \{ u_{nearest} \}$, $S \leftarrow S - \{ u_{nearest} \}$. Repeat this step until $OV(u_{nearest}, NUP_i) < d$.
 4. Start a new cluster $NUP_{i+1}$ and reiterate Step 2 and Step 3 until all rows in S are processed.
```

Figure 6. The *ROW-NEIGHBOR CLUSTERING* Algorithm.

After clustering, rows in $RW(v)$ are partitioned into several clusters NUP_1, NUP_2, …, NUP_k. Note that a cluster can be generated either by a NP or by a remote FPs. We use the following criteria to determine which is the case. If the number of rows in a cluster is no greater than a threshold d (normally 3), we shall conclude that this cluster is generated by remote FPs. Row entries in NUP_i which are FPs will be deleted. Since we assume the error rate of the false positive is from 0.6% to 2% (the total error rate for FPs and FNs together is from 3% to 10%), the probability that more than three FPs appear in the same column is very slim. The threshold value will be set according to the estimated FP rate of the data set, that is, the lower the FP rate the lower the threshold value. Since undetected FPs create catastrophic probe ordering, our threshold value is usually set higher in favor of the conclusion that a cluster is generated by FPs rather than by a NP. The trade-off is that we shall then have more probes deleted. The algorithm is summed up in Figure 7.

```
The NUP_RFP_SCREENING Algorithm.
Partition $RW(v)$ into clusters $NUP_1$, $NUP_2$, …, $NUP_k$ using the The ROW-NEIGHBOR CLUSTERING
    algorithm.
```

If a cluster NUP_i contains more than three rows, then we conclude that NUP_i is generated by a NP v. Delete rows in NUP_i from $RW(v)$ and generate a new column v_{NUPi} such that $RW(v_{NUPi}) = NUP_i$. Otherwise, columns in NUP_i are generated by remote FPs of rows in NUP_i. Eliminate these remote FPs by deleting entries in NUP_i.

Figure 7. The *NUP_RFP_SCREEING* Algorithm.

A complication can occur in step (1) above when we determine the clusters of $RW(v)$. Since it is possible for two nearby probes to form a NP, we might incorrectly place all rows in $RW(v)$ into the same cluster. However, such a NP normally does not create any serious problem since this NP column might generate several FPs and can be remedied by the FPs and FNs detection algorithm discussed later. Figure 8 is an example that the two clusters of $RW(v)$ be viewed as one cluster incorrectly.

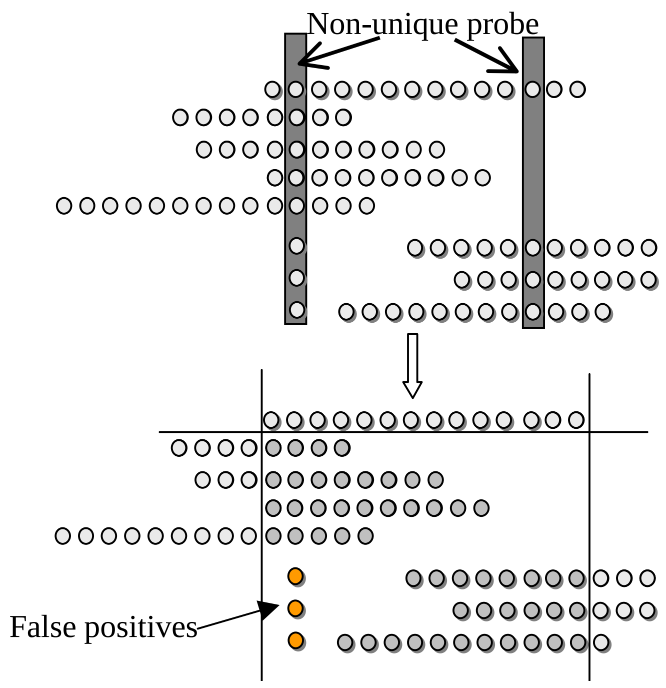

Figure 8. An example of two nearby probes being viewed as a non-unique probe

4.2. Screening out Chimeric Clones and Related Remote False Positives

The idea to screen out CCs is very similar to that of NP screening. Consider a CC u, which is formed by two or more normal clones. Figure 9 gives an example of a CC composed from two normal clones, where RC_1 and RC_2 denote the neighborhood of these two clones.

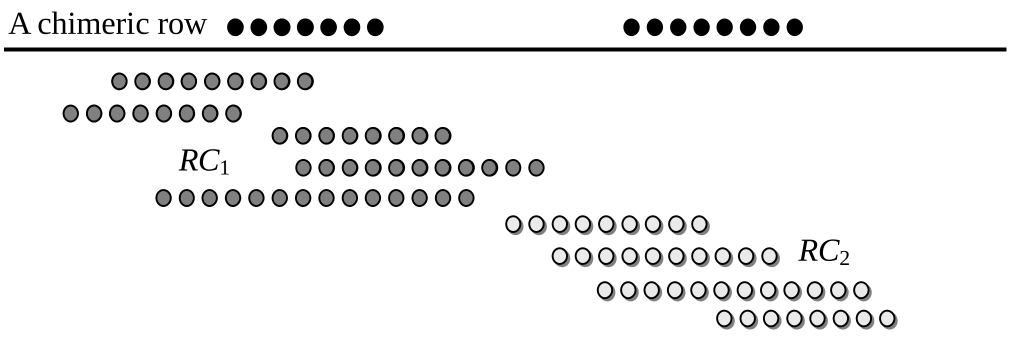

Figure 9. An example of a chimeric clone

Similar to the case of NPs, a cluster might be generated by a FP. Figure 10 gives an example of a cluster R_v generated by a remote FP.

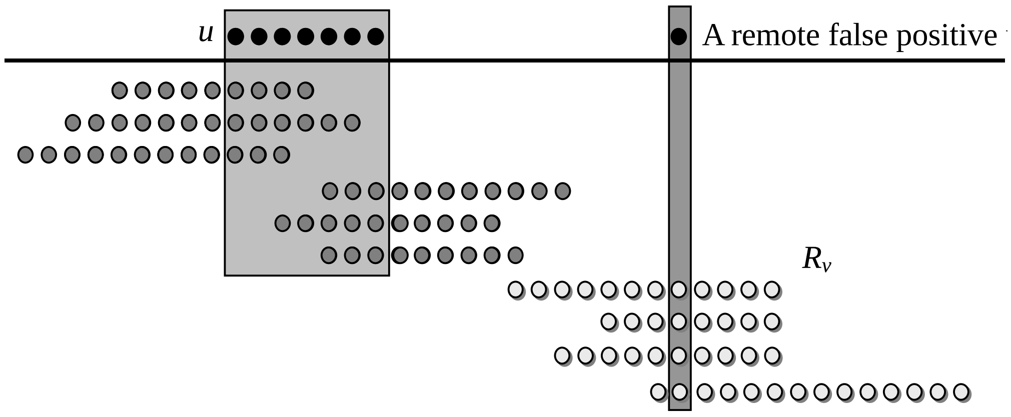

Figure 10. An example of a cluster R_v generated by a remote FP

We use the *ROW-NEIGHBOR CLUSTERING* algorithm described in the previous section to partition $RW(CL(u))$ into several clusters. Since our preliminary clustering strategy might over-split the $RW(CL(u))$ into more clusters than necessary, some clusters will be merged back. Figure 11 gives an example of two clusters that should be merged, where rows in cluster RC_i and rows in RC_j all overlap the same normal clone u.

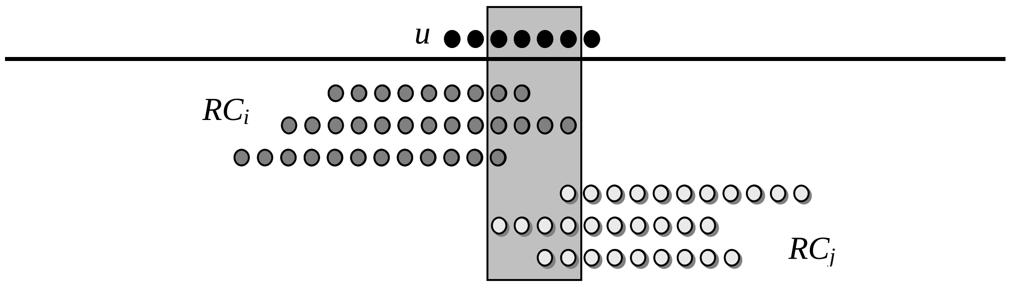

Figure 11. An example of two clusters that should be merged

Two clusters RC_a and RC_b will be merged if $| CL(RC_a) \cap CL(RC_b) \cap CL(u) | > 3$, Since we assume the error rate of FPs and FNs is normal than 5%, columns in $CL(RC_a)$

$\cap$ $CL(RC_b)$ $\cap$ $CL(u)$ might be consecutive with high probability. After this merging process, the clusters of $RW(CL(u))$ are fixed. Based on these clusters, we can now partition $CL(u)$ into subset CC_1, CC_2, …, CC_k such that v belong to CC_t, where RC_t is the cluster with the largest $|\{w|\ v\in CL(w)$ and $w\in RC_t\ \}|$. Similar to the NP screening, each subset of $CL(u)$ corresponds to either a normal clone segment or a set of remote FPs, and we use the following criteria to determine which is the case. If there are less than or equal to three columns in a subset, we shall conclude that the subset is a set of remote FPs. Since we assume the error rate of the FPs is at most 2%, the probability that more than three false positives appear in the same row is very slim. Note that the above threshold value can be modified for different data set as we discussed above. Column entries in CC_i that are FPs will be deleted. The overall CC and related remote FP screening algorithm is described in Figure 12.

```
The CC_RFP_SCREEING Algorithm.
1.  Partition $RW(CL(u))$ into clusters using the ROW-NEIGHBOR CLUSTERING algorithm.
2.  Merge two clusters $RC_a$ and $RC_b$ if $|\ CL(RC_a) \cap CL(RC_b) \cap CL(u)\ | > 3$.
3.  Partition $CL(u)$ into subsets $CC_1, CC_2, \ldots, CC_k$. If $RC_t$ is a cluster with the largest $|\{w|\ v\in CL(w)$
    and $w\in RC_t\ \}|$ value, then places $v$ into $CC_t$,
4.  If a subset $CC_i$ has more than three columns, then we regard $CC_i$ as a clone segment. Construct a
    new row $u_{CCi}$ with $CL(u_{CCi}) = \{v\ |\ v \in CC_i\ \}$. Otherwise, conclude that it is generated by remote FPs.
    Eliminate these remote FPs by deleting entries in $CC_i$.
```

Figure 12. The *CC_RFP_SCREEING* Algorithm.

Similar to the clustering of NP, a complication can occur in step (1) above when we determine the clusters of $RW(CL(u))$. Since it is possible that two nearby clones form a CC, we might incorrectly place all rows in $RW(CL(u))$ into the same cluster. However, such a mistake normally would not create a serious problem since this CC might generate several FNs and can be remedied by the FPs and FNs detection algorithm discussed later. Fig 13 gives such an example.

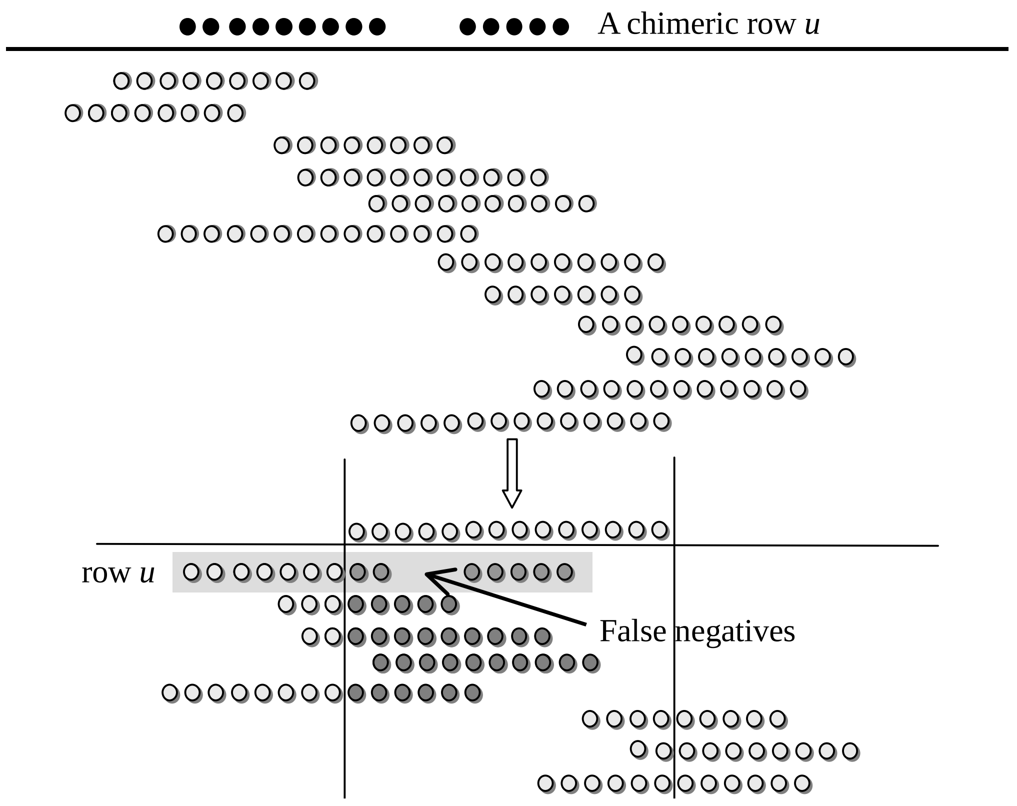

Fig 13. An example of two nearby clones being viewed as a chimeric clone

4.3. The Error-Tolerant Clustering of rows overlapping row *u*

The clustering algorithm discussed in this section forms the heart of our approach, which is based on neighborhood clustering. Assume there are no NPs, CCs or remote FPs, and we only have to deal with local FPs and FNs.

Consider the classification of rows overlapping row *u*. For a matrix *M* satisfying the *COP*, rows overlapping *u* can be partitioned into *A*(*u*), *B*(*u*) (as in the *COT* algorithm), *C*(*u*), *D*(*u*) and *I*(*u*) (as described in Figure 16). Such a classification can be carried out based on the overlapping relationships between those rows and *LF*(*u*), *RT*(*u*) (defined in lemma 3.4). Let *LL*(*u*) denote the set *LF*(*u*) ∩ *RW(CL*(*u*)), and *RR(u*) denote the set *RT*(*u*) ∩ *CL*(*RW(CL*(*u*))). That is, each row in *A*(*u*) should overlap with *LL*(*u*) only, each row in *B*(*u*) should overlap with *RR*(*u*) only. Any row overlapping both *LL*(*u*) and *RR*(*u*) must be in *C*(*u*), and any row overlapping none of *LL*(*u*) and *RR*(*u*) should be in *D*(*u*) or *I*(*u*). Figure 14 gives an example of *LL*(*u*) and *RR(u*).

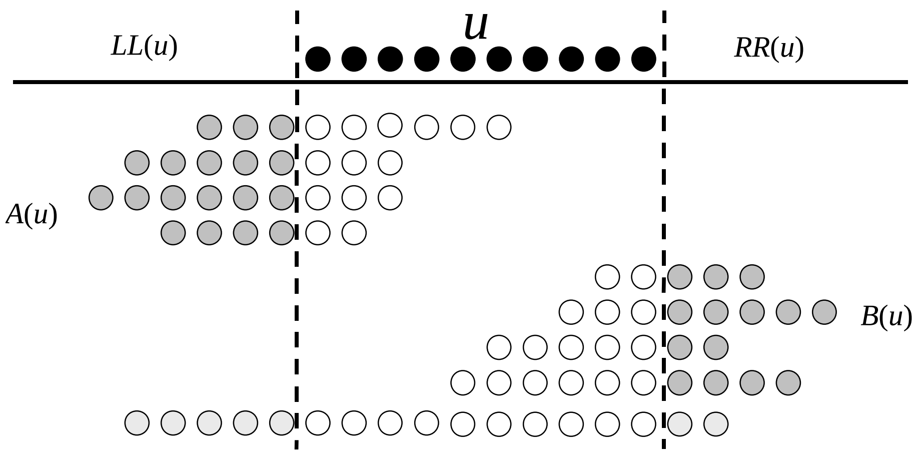

Figure 14. Example of *LL*(*u*) and *RR*(*u*)

The proof of the following lemma is similar to that of lemma 3.4.

Lemma 4.3.1. *If a* (*0,1*)*-matrix M satisfies the COP, then the collections* { *RW*(*v*) | *v*∈*LL*(*u*) } *and* { *RW(v)* | *v* ∈ *RR(u)* } *are monotone.*

In the event that the data contain errors, the classification of *LL*(*u*) and *RR(u)* could be obtained through a clustering analysis on *CL*(*RW(CL*(*u*))) - *CL*(u) based on Lemma 4.3.1. Let *d*(*u*,*v*) denote the Hamming distance between row *u* and row *v*, and *D*(*u*,*S*) = $\sum_{\forall v \in S} d(u,v) \big/ |S|$ denote the Hamming distance between row *u* and a set *S* of rows. The classification of *LL*(*u*) and *RR*(*u*) is described in Figure 15. Define $M_{STA(u)}$ to be the submatrix of *M* consisting of rows in *STA*(*u*) and columns in *CL*(*STA*(*u*))-*CL*(*u*). An example is illustrated by the shaded entries in Figure 14.

```
The LL-RR-CLASSIFICATION Algorithm
1. Denote the rows of $M_{STA(u)}$ by $R(M_{STA(u)})$ and the columns of $M_{STA(u)}$ by $C(M_{STA(u)})$
2. Let $b_1$ be the shortest column of $C(M_{STA(u)})$ and $b_2$ the shortest column among $C(M_{STA(u)})$ -
   $CL(RW(b_1))$. Let $LL(u) = \{b_1\}$ and $RR(u) = \{b_2\}$.
3. Classifying columns in $CL_{STA(u)} - \{b_1, b_2\}$ by Step 4 according the ascending ordering of their
   sizes.
4. If $D(v_i, LL(u)) < D(v_i, CLS_{RR(u)})$ then $LL(u) \leftarrow LL(u) \cup \{v_i\}$, otherwise $RR(u) \leftarrow RR(u) \cup \{v_i\}$.
```

Figure 15. The *LL-RR-CLASSIFICATION* Algorithm.

Due to FNs and FPs, some of the overlapping relationships between rows overlapping row *u, LL*(*u*) and *RR*(*u*) might be incorrect. We shall distinguish these errors as follows. For a column *w* in *A*(*u*), we should have *CL*(*w*)-*CL*(*u*) ⊆ *LL*(*u*) and *CL*(*u*)-*CL*(*w*) ≠ ∅. However, in case *CL*(*w*) ∩ *RR(u)* ≠∅, we conclude that entries in {(*w*,*v*) | *v* ∈ *CL*(*w*) ∩ *RR(u)*} are FPs. Similarly, for a column *w* in *B*(*u*), we should have *CL*(*w*)-*CL*(*u*) ⊆ *RT*(*u*) and *CL*(*u*)-*CL*(*w*) ≠ ∅. In case *CL*(*w*) ∩ *RR(u)* ≠ ∅, we conclude that entries in {(*w*,*v*) | *v*∈ *CL*(*w*) ∩ *RR(u)*} are FPs. For a column *w* in *C*(*u*),

$CL(u)$-$CL(w)$ should be empty. In case $CL(u)$-$CL(w) \neq \emptyset$, we conclude that entries in $\{(w,v) \mid v \in CL(u)\text{-}CL(w)\}$ are FNs. Such a classification scheme is summarized in Figure 16.

```
The NEIGHBOR-CLASSIFICATION Algorithm: Classify a row w.
 1. Calculate the error functions of row w as follows:
      $E_A(w) = |\{(w,v) \mid v \in CL(w) \cap RR(u)\}|$,
      $E_B(w) = |\{(w,v) \mid v \in CL(w) \cap LL(u)\}|$,
      $E_C(w) = |\{(w,v) \mid v \in CL(u) - CL(w)\}|$.
 2. Classify w into $A(u)$, $B(u)$, $C(u)$, $D(u)$ and $I(u)$ according to the following definitions:
      $A(u) = \{ w \mid CL(w)\text{-}CL(u) \neq \emptyset$ and $E_A(w) < E_B(w)$ and $E_A(w) < E_C(w)\}$,
      $B(u) = \{ w \mid CL(w)\text{-}CL(u) \neq \emptyset$ and $E_B(w) < E_A(w)$ and $E_B(w) < E_C(w)\}$,
      $C(u) = \{ w \mid CL(w)\text{-}CL(u) \neq \emptyset$ and $E_C(w) < E_A(w)$ and $E_C(w) < E_B(w)\}$,
      $D(u) = \{ w \mid CL(w) \subset CL(u) \}$,
      $I(u) = \{ w \mid CL(w) = CL(u) \}$ .
```

Figure 16 The *NEIGHBOR-CLASSIFICATION* Algorithm

Note that the determination of local FPs and FNs is a relative matter. Namely, one can change the phenomenon of a FP to that of a FN by changing the threshold value and there is a trade-off in deciding between the FPs and the FNs.

Finally, consider possible FNs within row u itself. Since FN entries of row u are likely column entries in the neighboring rows of u, they could be classified into $LL(u)$ or $RR(u)$. Without lost of generality, let v' be such a FN column in $LL(u)$. Consider the following two cases:

Case 1: $RW(v') \cap B(u) \neq \emptyset$.

In the *NEIGHBOR-CLASSIFICATION* algorithm, if $w \in RW(v') \cap B(u)$, the entry (w,v') will be considered as a FP of row w. However, if (u,v') is a FN, the entry (w,v') for $w \in RW(v') \cap B(u)$ might not have to be considered as a FP. We use some threshold value to make the decision. If $|RW(v') \cap B(u)|$ is greater than a threshold value, say 3, v' will be considered as a FN of row u with high probability. Otherwise, $w \in RW(v') \cap B(u)$ for $w \in RW(v') \cap B(u)$ will be still considered as FPs. For example, in Figure 17, the entry (u,v') will be considered as FN of row u itself since the number of the entries pointed by the arrow (FPs decided by *NEIGHBOR-CLASSIFICATION* algorithm) is greater than the threshold value.

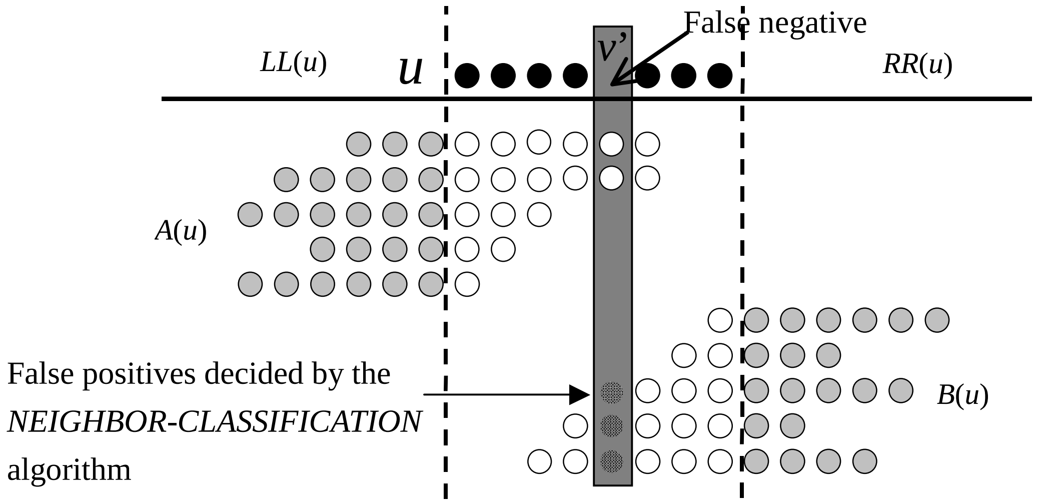
False negative
LL(u)
u
v'
RR(u)
A(u)
False positives decided by the NEIGHBOR-CLASSIFICATION algorithm
B(u)

Figure 17. Case 1 of the false negatives within row *u* itself

Case 2: *RW(v')* ∩ *B*(*u*) = ∅.

If *w** is a special row in *A*(*u*). The entries in {(*w*,*v'*)| *w* ∈ *RW(v')* ∩ *D*(*u*) and *CL*(*w*) ∩ *CL*(*w**) ≠ ∅} will be considered as FPs of row *w* due to *w**. Similarly, if (*u*,*v'*) is a FN, those (*w*,*v'*) entries might not have to be FPs. We also use a threshold value to make the decision. If the cardinality of |{(*w*,*v'*) | *w* ∈ *RW(v')* ∩ *D*(*u*) and *CL*(*w*) ∩ *CL*(*w**) ≠ ∅}| is greater than a threshold value, say 3, (*u*,*v'*) will be considered as a FN, otherwise the entries in {(*w*,*v'*)| *w* ∈ *RW(v')* ∩ *D*(*u*) and *CL*(*w*) ∩ *CL*(*w**) ≠ ∅} will be considered as FPs. For example, in Figure 18, the entry (*u*,*v'*) will be considered as FN of row *u* itself, since the number of the entries pointed by the arrow (FPs due to special row *u**) is greater than the threshold value.

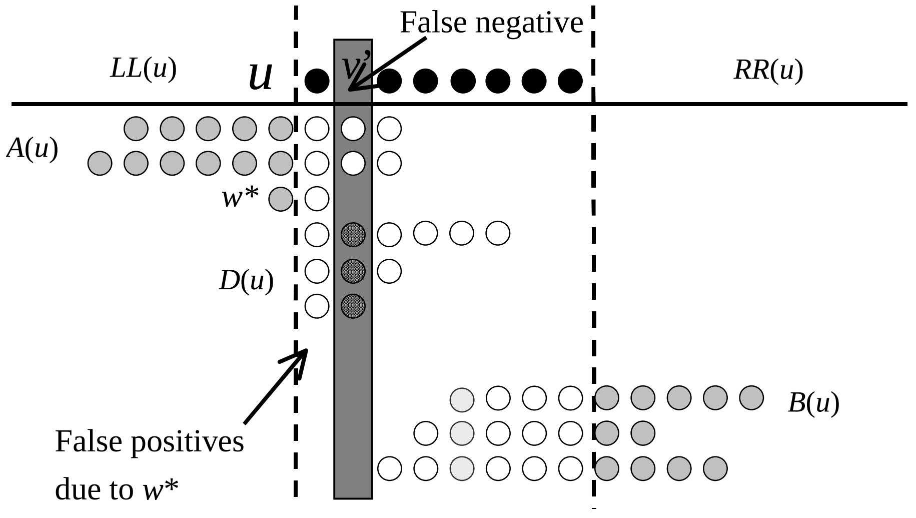

Figure 18. Case 2 of the false negatives within row *u* itself

4.4 Deciding Column Orderings under the Influence of Local False Negatives

and False Positives

In this section, we partition *CL*(*u*) into ordered sets that *induces* a collection of column orderings under local FNs and FPs. The monotone collections related to *A*(*u*) and *B*(*u*) both inside the set *CL*(*u*) and outside it provide a very strong structural property for matrices satisfying the *COP*. This structure is stable enough for us to obtain a "good" column partition of *CL*(*u*) even when the input matrix contains errors.

In case there are a few 1's missing from the input data (FNs) in *A*(*u*) or *B*(*u*), they can be inferred from the monotone structure. For example, in Figure 19, the collections of sets $\{A(u) \cap RW(v') \mid v' \in CL(u)\}$ do not satisfy the monotone property. However, they will if the 0-entry pointed by the arrow is changed to 1. Thus, this particular 0-entry can be inferred to be a FN by the monotone property. Similarly, if there are a few 1's in *A*(*u*) or *B*(*u*) which should have been 0's, then they can be inferred to as FPs. For the same example in Figure 19, when the 1-entry pointed by the arrow is changed to 0, the collection $\{B(u) \cap RW(v') \mid v' \in CL(u)\}$ will satisfy the monotone property.

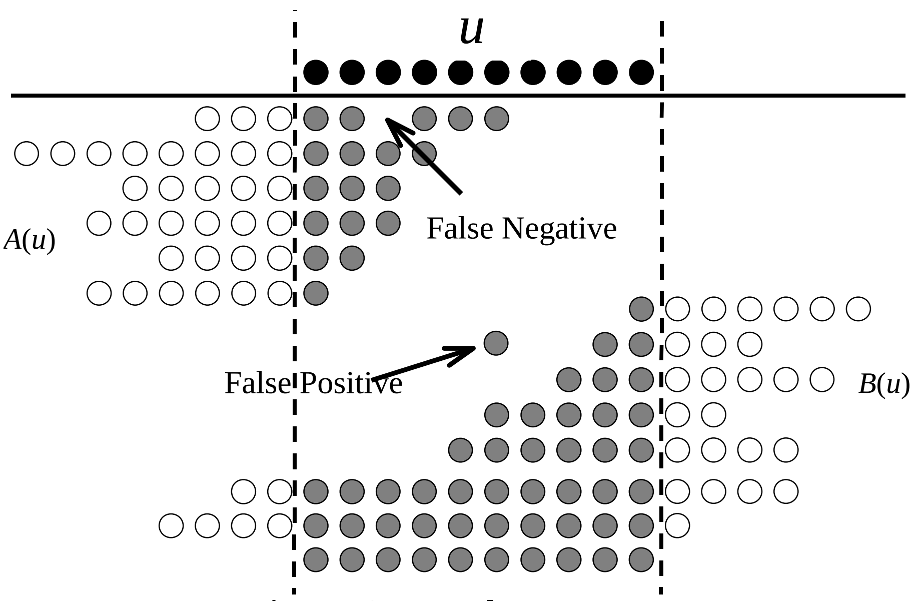

Figure 19. FP and FN errors

In general, if a collection of sets does not satisfy the monotone property, we could remove some elements and add some other elements into sets in the collection so that the monotone property is satisfied. We denote the removed elements as *removals* and the added elements as *fill-ins*. The removals could be considered as FPs "determined" by the system, and the fill-ins could be considered as FNs. Hence, one objective of removing FPs and FNs is to minimize the total number of fill-ins and removals needed to modify $\{ A(u) \cap RW(v) \mid v \in CL(u) \}$ and $\{B(u) \cap RW(v) \mid v \in CL(u)\}$ to satisfy the monotone property simultaneously. Note that the minimum fill-in problem is NP-complete [Yannakakis 1981] and a polynomial approximation algorithm for this problem with at most eight times the optimum have been proposed in [Natanzon et al. 1998]. However, it is not known whether finding the minimum total number of fill-ins and removals is still NP-hard. In order to deal with the compatibility of sub-rows in $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$ and $\{ CL(w) \cap CL(u) \mid w \in B(u) \}$in a more concise fashion, we shall consider the following collection of sets $S(u) = \{ RW_1(v) \mid v \in CL(u)\}$

Definition 4.4.1. *For each column v in CL(u), define the special set of row indices $RW_1(v)$ associated with u to be $(A(u) \cap RW(v)) \cup (B(u) - RW(v))$, where A(u), B(u) and RW(v) are considered as sets of row indices rather than sets of rows.*

An example of a $RW_1(v)$ is shown in Figure 20. Note that each $RW_1(v)$ is not a sub-column of M; rather, it could be regarded as an “artificial column”. Since $B(u) - RW(v)$ and $B(u) \cap RW(v)$ is a partition of the set of row indices in $B(u)$, $B(u) - RW(v)$ is the complement of $B(u) \cap RW(v)$ for column v. Thus, for each row $w \in A(u)$, $w \in RW_1(v)$ iff $w \in RW(v)$, and for each $w \in B(u)$, $w \in RW_1(v)$ iff $w \notin RW(v)$. The purpose of introducing $S(u) = \{ RW_1(v) \mid v \in CL(u)\}$ is that, originally, we need to check the monotonicity of two sets of sub-rows, $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$, $\{ CL(w) \cap CL(u) \mid w \in B(u) \}$ as well as the compatibility of these two sets; but now, these can all be reduced to checking the monotonicity of the set $S(u)$ as proved below.

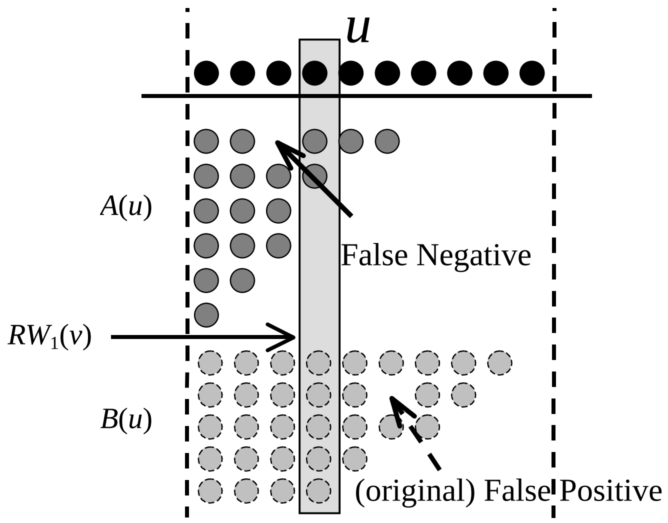

Figure 20. An example of { $RW_1(v) \mid v \in CL(u)$}.

Lemma 4.4.2. $S(u) = \{ RW_1(v) \mid v \in CL(u)\}$ *is monotone iff*

1. *the two sets of sub-rows* { *$CL(w) \cap CL(u) \mid w \in A(u)$* }, { *$CL(w) \cap CL(u) \mid w \in B(u)$* } *are monotone and $CL(u)$ can be uniquely partitioned with v^A, v^B placed at one end, respectively.*
2. *Each row in $A(u)$ is compatible with the column partition determined by* { *$CL(w) \cap CL(u) \mid w \in B(u)$* }, *and each row in $B(u)$ is compatible with the column partition determined by* { *$CL(w) \cap CL(u) \mid w \in A(u)$* }.

Proof. We first show the "only if" part. Assume $S(u)$ is monotone. Then for any two columns v_1, v_2 in $CL(u)$, either $RW_1(v_1) \supseteq RW_1(v_2)$ or vice versa. Without loss of generality, assume $RW_1(v_1) \supseteq RW_1(v_2)$. Since $RW_1(v)$ is the disjoint union of $(A(u) \cap RW(v))$ and $(B(u) - RW(v))$ for any v, We must have $A(u) \cap RW(v_1) \supseteq A(u) \cap RW(v_2)$ and $B(u) - RW(v_1) \supseteq B(u) - RW(v_2)$. The former implies that columns in $\{ A(u) \cap RW(v) \mid v \in CL(u) \}$ are monotone. The latter implies that $B(u) \cap RW(v_1) \subseteq B(u) \cap RW(v_2)$, and thus, columns in $\{ B(u) \cap RW(v) \mid v \in CL(u) \}$ are monotone. From Lemma 3.2, sub-rows in $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$ and those in $\{ CL(w) \cap CL(u) \mid w \in B(u) \}$ are monotone.

The monotone property of $S(u)$ implies that elements in the collection can determine a unique partition of $CL(u)$. Let $S_1, S_2, .., S_d$ be such a partition that for any two columns $v_1 \in S_i$, $v_2 \in S_j$, $RW_1(v_1) \supseteq RW_1(v_2)$, if $i < j$. We have $A(u) \cap RW(v_1) \supseteq A(u) \cap RW(v_2)$. Recall that v^A is a column in $CL(A(u)) \cap CL(u)$ with the largest $|RW(v^A) \cap A(u)|$, thus $v^A \in S_1$ and v^A can be placed at one end of $CL(u)$. Similarly, v^B can be placed at the other end of $CL(u)$ according the monotone property of $\{ B(u) \cap RW(v) \mid v \in CL(u) \}$.

We now show that every row in $B(u)$ is compatible with the column partition determined by $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$. Consider a row w in $B(u)$. Since rows w and u are strictly overlapping, $CL(w) - [S_1 \cup S_2 \cup ... \cup S_d] \neq \emptyset$. Let S_s be a set having nonempty intersection with $CL(w)$. For any set S_t with $t > s$, and any two columns $v_1 \in S_s$, $v_2 \in S_t$, $B(u) \cap RW(v_1) \subseteq B(u) \cap RW(v_2)$. That is, all sets to the right of S_s are contained in $CL(w)$. We have shown that every row in $B(u)$ is compatible with the column partition determined by $S(u)$. Since the column partition determined by $S(u)$ is a refinement of that determined by $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$, we have proved that every row in $B(u)$ is compatible with the column partition determined by { $CL(w)$

$\cap$ $CL(u)$ | $w \in A(u)$ }. The proof that each row in $A(u)$ is compatible with the column partition determined by { $CL(w) \cap CL(u)$ | $w \in B(u)$ } is symmetric.

Finally, consider the "if" part. Recall that we can get a unique partition of $CL(A(u)) \cap CL(u)$ based on the monotone collection { $CL(w) \cap CL(u)$ | $w \in A(u)$ } in which v^A is at one end (this forces a unique partition). And we can form a unique compatible partition of $CL(B(u)) \cap CL(u)$ with v^B at the other end. Refine these two partitions based on their overlapping (compatible) rows, we can get a unique partition $S_1, S_2,.., S_d$ of $CL(u)$ such that $v_1 \in S_i$, $v_2 \in S_j$, $A(u) \cap RW(v_1) \supseteq A(u) \cap RW(v_2)$ and $B(u) \cap RW(v_1) \subseteq B(u) \cap RW(v_2)$, if $i < j$. Thus, $RW_1(v_1) \supseteq RW_1(v_2)$, and $S(u)$ satisfies the monotone property. █

From lemma 4.4.2, eliminating FPs and FNs can be modeled as minimizing the total number of fill-ins and removals such that the collection { $RW_1(v)$ | $v \in CL(u)$ } after modification satisfies the monotone property. Note that, for $w \in A(u)$, the fill-ins could be considered as FNs and the removals could be considered as FPs , and for $w \in B(u)$, the fill-ins could be considered as FPs and the removals could be considered as FNs. Recall that removals and fill-ins are relative to each other, and there is a trade-off in determining FPs and FNs. For example, we can allow a "one" to "stay" by filling in the "missing" elements or we can remove a "one" without filling in any "missing" element in the corresponding rows (to satisfy the monotone property).

Our strategy is to detect potential FPs of $S(u)$ (FPs for $A(u)$ and FNs for $B(u)$) first; remove them and then deal with the fill-ins. A "one" is considered as a FP based on the following heuristic. Let $RW_1(v_1)$, $RW_1(v_2)$, … , $RW_1(v_{|CL(v)|})$ be a list ordered according to their ascending sizes. If { $RW_1(v)$ | $v \in CL(u)$ } satisfies the monotone property, then we should have $RW_1(v_i) \subseteq RW_1(v_j)$ for all $i < j$. However, since the input data contain errors, $RW_1(v_i)$ might not be contained in every such $RW_1(v_j)$. But, since the error rate of FNs and FPs is no more than 10%, we expect that $RW_1(v_i)$ is contained in $RW_1(v_j)$ with high probability. For each $w \in RW_1(v_i)$, if |{ j | $w \notin RW_1(v_j)$ for all $i < j$ }| ≥ 3, the entry (w, v_i) is considered as a FP, since the probability that there are more than three fill-ins in the same row is relatively low.

We now determine the fill-ins to make the collection of sets { $RW_1(v)$ | $v \in CL(u)$ } satisfy the monotone property. For each column v in $CL(u)$, define the function, *fill-in*(v), as the set { (w,v') | $w \in RW_1(v)$ and $w \notin RW_1(v')$ for all $v \neq v'$ }, which is the set of elements one needs to fill so that $RW_1(v') \supseteq RW_1(v)$ for every other column v'. The algorithm for detecting FPs and FNs is summarized in Figure 21.

The *FP-FN- DETECTION* Algorithm

1. Let $RW_1(v) = (A(u) \cap RW(v)) \cup (B(u) - RW(v))$.
2. Sort columns in $CL(u)$ into a list $\{v_1, v_2, \ldots, v_{|CL(u)|}\}$ according to their ascending $|RW_1(v)|$ values.
3. For each $w \in RW_1(v_i)$, if $|\{ w \mid w \notin RW_1(v_j) \text{ for all } i < j \}| \geq 3$, then (w, v_i) is considered as a removal. If $w \in A(u)$, remove w from $RW(v_i)$, otherwise, add w to $RW(v_i)$.
4. Select a column v^* of $CL(u)$ with the minimum *fill-in*(v^*). Then element (w,v') in *fill-in*(v^*) is considered as a fill-in. If $w \in A(u)$, add w to $RW(v')$, otherwise, remove w from $RW(v')$. Remove v^* from $CL(u)$. Reiterate Step 4 until all columns in $CL(u)$ have been processed.

Figure 21. The *FP-FN-DETECTION* Algorithm

Once $\{ RW_1(v) \mid v \in CL(u)\}$ satisfies the monotone property, we can partition columns in $CL(u)$ into a unique partition based on the monotone collection $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$, in which v^A is at one end of the partition; similarly, we can partition columns in $CL(u)$ into a unique partition based on the monotone collection $\{ CL(w) \cap CL(u) \mid w \in B(u) \}$, in which v^B is at one end. Refine these two partitions based on their mutually overlapping compatible rows. Now, further refine this partition by bringing in compatible rows in $D(u)$ one by one. This gives the final internal partition of $CL(u)$.

Furthermore, for each column v, the number of removals and fill-ins generated by the *FP-FN-REMOVE* Algorithm provides a good measure for the quality of this column. The FN error rate, $FN(v)$, of column v could be estimated by number of fill-ins divided by (number of ones in column v originally - number of removes + number of fill-ins) and the FP error rate, $FP(v)$, of column v could be estimated by number of removals divided by (number of zeros in column v originally - number of fill-ins + number of removes). We rate the confidence of position of column v in four levels as follows. If $FN(v) < 3\%$ and $FP(v) < 1\%$, mark the confidence level to be 4 meaning the column information is “reliable”; if $FN(v) < 8\%$ and $FP(v) < 1\%$, mark the confidence level to be 3 meaning “believable”; if $FN(v) < 16\%$ and $FP(v) < 1\%$, mark the confidence level to be 2 meaning “doubtful”; otherwise, mark the confidence level to be 1 meaning “problematic”. Thus, high confidence signifies that the information of column v is relatively reliable (so does its predicted position), and low confidence signals that there is a potential problem (for the biologists) in this column.

When it appears that a probe creates a disrupting behavior for its neighboring clones, this probe could be deleted from further consideration. Thus, rather than forcing all probes to be included and ordered in the final arrangement, our algorithm could produce more than one contig.

4.5. A Summary of the Error-Tolerant Algorithm for the *COT*

Finally, we summarize the above algorithms for dealing with all four types of errors in this subsection. Since different error types affect our algorithm to different extent, we need to eliminate them in different stages. An error type having a far-reaching effect will be eliminated earlier to reduce its potential disruption. Our error-tolerant *COT* algorithm involves the following three stages:

(1) For each probe determine whether it is a NP. Separate (or discard) a NP and eliminate remote FPs discovered along the way;

(2) For each clone determine whether it is chimerical. Separate (or discard) CCs and eliminate remote FPs discovered along the way;

(3) Decide the column ordering under the influence of FPs and FNs. These are described in Figure 22.

The *Error-Tolerant COT* Algorithm

Stage 1: For each probe determine whether it is a NP. Separate (or discard) a NP and eliminate remote FPs discovered along the way using the *NUP_RFP_SCREEING* Algorithm in Section 4.1.

Stage 2: For each clone determine whether it is chimeric. Separate (or discard) CCs and eliminate remote FPs discovered along the way using the *CC_RFP_SCREEING* algorithm in Section 4.2.

Stage 3: Decide the column ordering under the influence of FPs and FNs as follows. Process the rows according to an ascending order of their sizes. The main iteration is described below.

1. If $|CL(u)| \leq 1$, delete u. Proceed to the next row.
2. Construct $LL(u)$ and $RR(u)$ using the *LL-RR-CLASSIFICATION* algorithm.
3. Partition $RW(CL(u))$ into $A(u)$, $B(u)$, $C(u)$, $D(u)$ and $I(u)$ using the *NEIGHOBR-CLASSIFICATION* algorithm.
4. Let v^A be a column in $CL(A(u)) \cap CL(u)$ with the largest $|RW(v^A) \cap A(u)|$. Let v^B be a column in $CL(B(u)) \cap CL(u)$ with the largest $|RW(v^B) \cap B(u)|$.
5. Remove FPs and FNs using the *FP-FN-REMOVE* algorithm.
6. Partition $CL(u)$ using sub-rows in the three sets $\{ CL(w) \cap CL(u) \mid w \in A(u) \}$, $\{ CL(w) \cap CL(u) \mid w \in B(u) \}$ and $\{ CL(w) \cap CL(u) \mid w \in D(u) \}$ to obtain a unique partition.
7. Calculate the confidence level for each column of $CL(u)$.
8. Delete all columns in $CL(u)$ except v^A and v^B. Construct a new special row u^S with $CL(u^S)=\{v^A,v^B\}$. Proceed to the next row.

Figure 22. The *Error-Tolerant COT* Algorithm

5. Computational Results

We conduct experiments on both synthetic data and real genomic data using our error-tolerant *COT* algorithm.

5.1. Results on Synthetic Data

In the experiment on synthetic data, we start with matrices satisfying the *COP* and randomly create errors. Then feed the resultant matrices to our algorithm to get a final column ordering. The evaluation is based on the deviation of the new ordering produced by the algorithm from the original one. We use three fixed matrices of sizes 100x100, 200x200, and 400x400 that satisfy the *COP*. These matrices are generated randomly under the constraint that the number of 1s in each row ranges from 5 to 15. The errors are created as follows.

1. The error rates for NPs and CCs are generated at three different levels, 0%, 1% and 2%, respectively, to observe how the algorithm cope these errors at different levels. To generate a NP, we merge two columns into one. To generate a CC, we merge two rows into one. For example, for a 100x100 matrix, we shall generate 1 NP, 1 CC at the error rate 1%, and 2 NPs, 2 CCs at the error rate 2%. For a 400x400 matrix, we shall generate 4 NPs, 4 CCs at the error rate 1%, and 8 NPs, 8 CCs at the error rate 2%.
2. On top of the errors of NPs and CCs, we shall generate additional (combined) errors of FPs and FNs at 3%, 5% and 10% rates, respectively. Within each error percentage, the ratio of the number of FPs and that of FNs is set to be 1 to 4, namely, for every FP generated, there will be 4 false negatives generated. For a 100x100 matrix, let the total number of 1s be *k*. At the error rate of 3%, we shall generate 0.006k FPs by randomly changing 0.006k 0-entries (among all 0-entries) to 1s. Similarly, we shall generate 0.024*k* FNs by randomly changing 0.024*k* 1-entries (among all 1-entries) to 0s. For a 400x400 matrix, let the total number of 1s be *q*. At the error rate of 10%, we shall generate 0.02*q* FPs by randomly changing 0.02 *q* 0-entries to 1s. Similarly, we shall generate 0.08*q* FNs by randomly changing 0.08*q* 1-entries to 0s.

For each matrix at different sizes, we generate errors at different levels, namely NPs and CCs at 0%, 1%, 2% and the total number of FPs and FNs at 3%, 5% and

10%. For each error combination generated, we repeat the experiment 50 times based on different random seeds. The results are evaluated by comparing the resultant column ordering from that of the original ordering using the measures defined below.

Definition 5.1.1. *For a column v, let d_1 be the number of columns ordered to the left of v but whose indices are greater than v and d_2, the number of columns ordered to the right of v whose indices are less than v. Let the* ***displacement*** *d(v) of column v be the larger of d_1 and d_2.*

The displacement *d*(*v*) gives an approximate measure of the distance of column *v* from its "correct" position. It should be noted that an exact measure is difficult to define here since many other columns have to be moved simultaneously in order for column *v* to be placed correctly. We have the following three criterion for measuring the total deviation of the resultant ordering from the original one:

Definition 5.1.2. *The* ***average displacement*** *of a column ordering is the average of the displacement of all columns in the resultant order.*

Definition 5.1.3. *If the displacement of a column v is more than 4, we say v is a* ***jump column****. The* ***jump percentage*** *is the number of jump columns divided by the total number of columns.*

Definition 5.1.4. *The* ***average difference*** *of the column ordering is the average of the difference in the column indices of adjacent columns in the resultant order.*

For example, in Figure 23, *d*(2) = 6 (there are 6 columns ordered to the left of column 2 whose indices are greater than 2), *d*(6) = 1, and *d*(8) = 6 (there are 6 columns ordered to the right of column 8 whose indices are less than 8). Thus, column 2 and column 8 are jump columns. The average displacement is 1.7, and the average difference is 3.2. Note that for a perfect ordering, the average difference is 1.

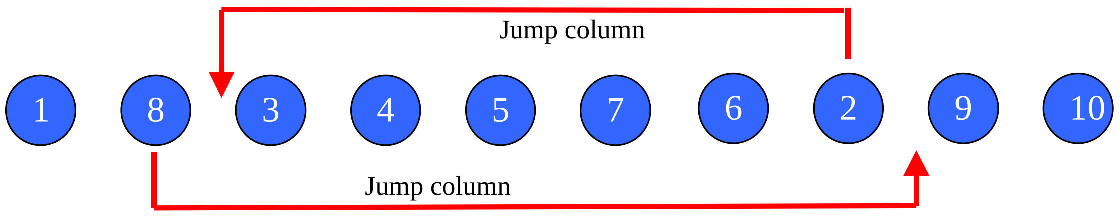

Figure 23. An example of jump columns.

We shall measure the performance of our algorithm by counting (1) the number of jump columns; (2) the number of average displacement and (3) the number of average difference in the resultant order. The philosophy of using the number of jump columns as a measure is that, it is the larger column displacement we want to avoid rather the smaller one. In case there is a big block of columns misplaced, then every column in that block could be a jump column. So there is a big penalty for block misplacement (sometimes, such a penalty is doubled). This assumption seems to be acceptable for those biologists we have consulted.

The total displacement is greater than or equal to the total number of column swaps required to place the columns in increasing order. This can argued as follows. For the column v^n with the largest index, we can move it to the rightmost position without using more than $d(v^n)$ swaps. Then the column v^{n-1} with the second largest index can be moved to the left of column v^* without using more than $d(v^{n-1})$ swaps. The rest can be argued recursively. Thus, the average displacement reflects the average behavior of displacement of all columns.

Now, the third measure, the average difference, does not penalize the block displacement as much as in the previous two measures. But, it penalizes columns deleted by our algorithm.

In Figures 24, 25 and 26, we plot the curve of the accumulative number of matrices among the 50 (the y-axis) matrices against the jump percentage (the x-axis) at different FP and FN error rates, 3%, 5% and 10%, respectively. Furthermore, we assume the largest error rate, 2%, for NP and CCs in all three cases. In case the final probe ordering are broken into several islands (because of column deletion), the jump percentages are calculated for each island separately. As one can see, even when the FP and FN error rate is 10%, the percentage of jump columns in most cases is still less than 5%. This indicates that the final probe ordering produced by the algorithm is a good approximation for the original. In a few bad instances where the jump percentages are over 15%, the errors are often caused by the incorrect order of two large blocks within an island. More detailed statistics for the jump percentage are listed in Table 1 to Table 3. Table 4 and Table 5 list the average number of displacement and the average difference separately. In most cases, the average number of displacement is less than 0.5, and the average number of difference is less than 1.6 (again, for a perfect ordering, the average difference is 1). Table 6 and Table 7 list the average number of islands and the percentage of deleted columns separately. Even when the error rate of FPs and FNs is 10%, the average number of islands is still about 6, and in most cases, the percentage of deleted columns is less than 10%.

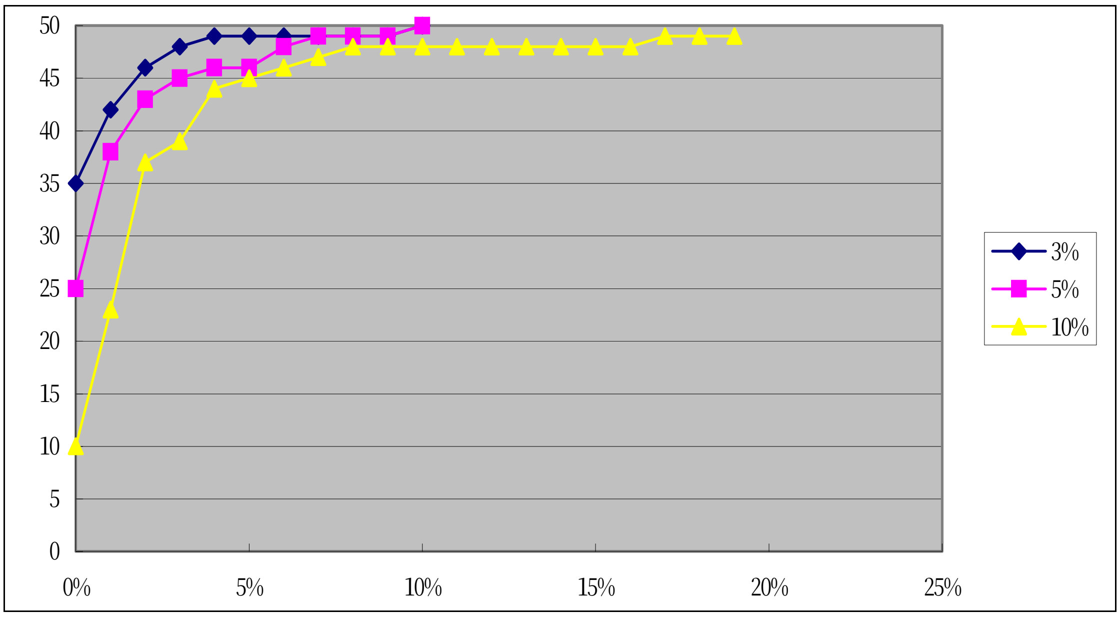

Figure 24. The result of 5-jump percentages running fifty 100x100 matrices with 2% NPs and CCs.

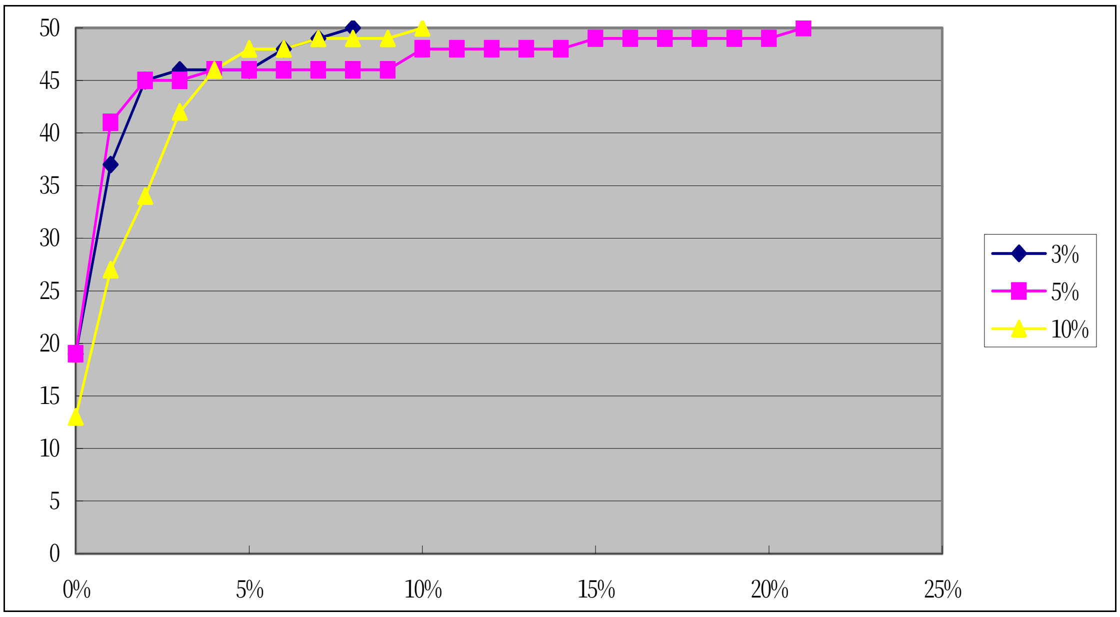

Figure 25. The result of 5-jump percentages running fifty 200x200 matrices with 2% NPs and CCs

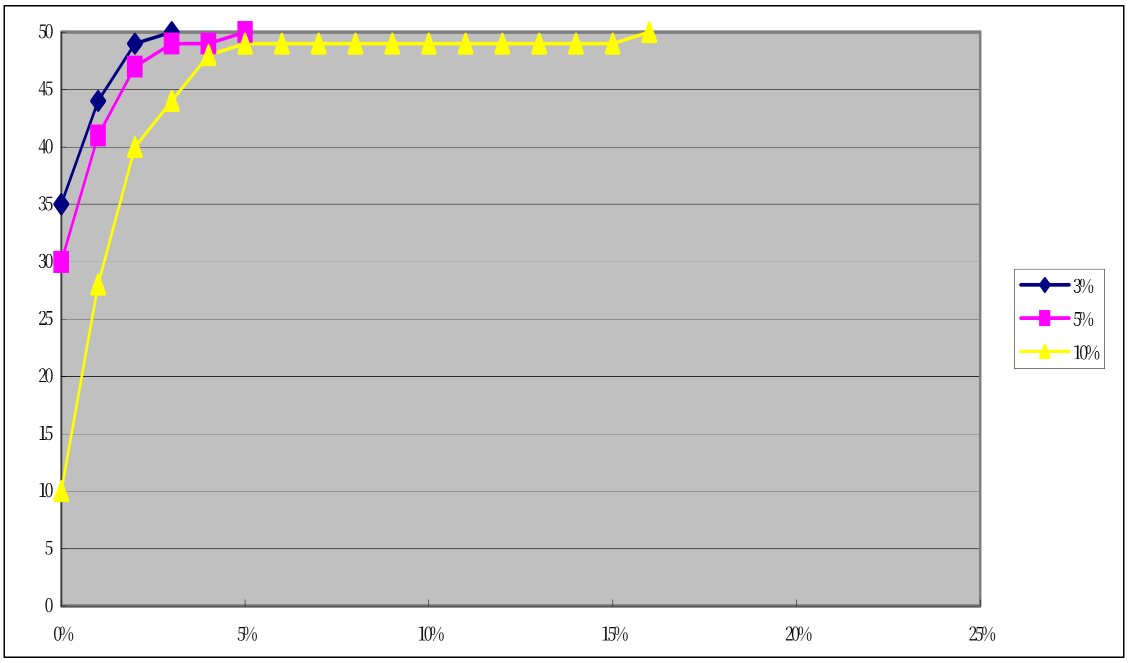

Figure 26. The result of 5-jump percentages running fifty 400x400 matrices with 2% NP and CCs

Table 1. Accumulative percentage of matrices whose columns are within the jump percentage (without CC and NP).

Jump percentage	100x100 matrix			200x200 matrix			400x400 matrix		
	3%	5%	10%	3%	5%	10%	3%	5%	10%
0	58%	54%	18%	62%	30%	10%	78%	42%	14%
5	100%	96%	88%	96%	88%	92%	100%	100%	90%
10	〃	98%	94%	98%	98%	98%	〃	〃	100%
15	〃	98%	98%	98%	100%	100%	〃	〃	〃
20	〃	100%	100%	100%	〃	〃	〃	〃	〃

Table 2. Accumulative percentage of matrices whose columns are within the jump percentage (with 1% NPs and CCs).

Jump percentage	100x100 matrix			200x200 matrix			400x400 matrix		
	3%	5%	10%	3%	5%	10%	3%	5%	10%
0	56%	46%	30%	38%	30%	22%	68%	60%	18%
5	94%	100%	92%	94%	88%	86%	100%	100%	98%
10	98%	〃	92%	100%	100%	94%	〃	〃	100%
15	100%	〃	96%	〃	〃	98%	〃	〃	〃
20	〃	〃	100%	〃	〃	100%	〃	〃	〃

Table 3. Accumulative percentage of matrices whose columns are within the jump percentage (with 2% NPs and CCs).

Jump percentage	100x100 matrix			200x200 matrix			400x400 matrix		
	3%	5%	10%	3%	5%	10%	3%	5%	10%
0	70%	50%	20%	38%	38%	26%	70%	60%	20%
5	98%	92%	90%	92%	92%	96%	100%	100%	98%
10	100%	100%	96%	100%	96%	100%	〃	〃	98%
15	〃	〃	96%	〃	98%	〃	〃	〃	98%
20	〃	〃	100%	〃	98%	〃	〃	〃	100%
25	〃	〃		〃	100%	〃	〃	〃	〃

Table 4. Average number of displacement.

Error rate of CC and NUP	0%			1%			2%		
Error rate of FP and FN	3%	5%	10%	3%	5%	10%	3%	5%	10%
100x100 matrix	0.02	0.08	0.30	0.03	0.19	0.28	0.02	0.06	0.37
200x200 matrix	0.04	0.12	0.24	0.05	0.06	0.16	0.05	0.44	0.77
400x400 matrix	0.06	0.16	0.56	0.09	0.23	0.39	0.92	0.82	1.01

Table 5. Average difference of indices of adjacent columns.

Error rate of CC and NUP	0%			1%			2%		
Error rate of FP and FN	3%	5%	10%	3%	5%	10%	3%	5%	10%
100x100 matrix	1.34	1.52	1.91	1.37	1.57	1.90	1.39	1.60	1.97
200x200 matrix	1.44	1.61	2.09	1.51	1.64	2.22	1.56	1.68	2.07
400x400 matrix	1.36	1.54	1.94	1.38	1.56	1.97	1.46	1.56	1.99

Table 6. Average number of islands.

Error rate of CC and NUP	0%			1%			2%		
Error rate of FP and FN	3%	5%	10%	3%	5%	10%	3%	5%	10%
100x100 matrix	1.04	1.16	1.64	1.1	1.26	1.88	1.12	1.28	2.06
200x200 matrix	1.68	2.1	3.72	1.76	2.26	4.16	2.14	2.26	3.62
400x400 matrix	1.7	2.62	6.22	1.74	3.4	6.06	1.9	3.04	6.22

Table 7. Percentage of deleted columns.

Error rate of CC and NUP	0%			1%			2%		
Error rate of FP and FN	3%	5%	10%	3%	5%	10%	3%	5%	10%
100x100 matrix	2.76	4.24	8.88	2.8	4.36	10.5	2.54	4.29	10.7
200x200 matrix	3.37	5.68	12	3.58	5.39	12.7	3.88	6.58	13
400x400 matrix	3.47	5.97	13.5	3.49	6.21	14.3	3.63	6.09	16.7

Next, we give two outputs from our COT program in Figures 27 and 28. These matrices do not contain NPs or CCs. In Figure 27, we give the result of a 50x50 matrix at error rate 5%. The matrix on the left is generated from the original matrix whose positions of “1” and “0” are represented by “1” and dot, respectively. The FNs and FPs generated from the original matrix are represented by “N”s and “P”s, respectively. The matrix on the right is the resultant matrix generated by our program in which “F” represents the fill-in elements. The number at the top of each column of the resultant matrix is the confidence level of this column.

In this example, we get the ordering of columns of the resultant matrix from left to right as follows:

1 2 3 4 6 7 8 10 9 5 11 12 13 14 15 16 17 18 19 20 21 23 24 25 26 27 28 30 29 31 32 33 34 35 36 37 39 40 41 42 43 44 45 46 47 49 50.

In the above column ordering, column 5 is a 5-jump. Columns 22, 38, 48 are deleted. The number of average displacement is 0.19 and the number of average difference is 1.3.

In Figure 28, we give the result of a of a 50x50 matrix at the higher error rate, 10%. In this example, the resultant column ordering from left to right is as follows:

1 5 17 2 3 4 6 7 8 9 12 16 14 15 13 18 10 19 21 22 23 24 25 26 27 28 30 29 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 48 47 49.

In the above column ordering, columns 17 and 10 are jump columns. Columns 11, 20, 50 are deleted. The number of average displacement is 0.9 and the number of average difference is 2.23.

```
                                                        44442444414444344443334424444444424344444444244
11111N1111111111N11...............................      1111F11111111111F11F...........................
.11111111111......................................      .11111111111...................................
...1111111111.....P...............................      ...1111111111..................................
...11111111111N1111111............................      ...11111111111F111111..........................
...1111111111111111111...............P............      ...111111111111111111..........................
.....1111111111...................................      ....11111F11111................................
.....111111111111111111...........................      ....11111F111111111111.........................
.......N11111111111...............................      .......11F11111111.............................
.......1111111111.................................      ......111F1111111..............................
........111111111111N111..........................      .......11F1111111111F11........................
........1N1111111111111111........................      ........1F111111111111111......................
..........111111111111111111...................P..      ..........11111111111111111....................
....P.....111111..................................      .........1111111...............................
...........11111111111111.........................      ...........1111111111111.......................
...........1N111111111111N1N......................      .............11111111111F1.....................
...........1111111111111111111....................      ...........111111111111111111..................
............1111N11111111111N1....................      ............1111F11111111111...................
............111111111111111.......................      ............11111111111111.....................
.................1111111111111N11.................      .................111111111111..................
..................1111N11111111111................      ..................111F11111111111..............
..................11111111........................      ..................1111111......................
..................1111111111111...................      ..................11111111111..................
...................111111.........................      ...................11111.......................
....................1111111N111111................      ....................1111FFF111111..............
......................11111111....................      .....................11111111..................
........................1111N1....................      .......................11111...................
..........................P11N11111111............      ............................11111111...........
...........................11111111111111.........      ..........................1111111111111........
.............................11111111111111111....      ...........................1FF11111111111111...
..............................11111111111111......      ..............................111111111111.....
.....................P.........11111111...........      ..............................1111111..........
................................111111111111111111      ...............................1111111111111111
.................................1111111111111....      ................................111111111111...
.................................11111111111111111      ................................111111111111111
.................................1111111111.......      ................................111111111......
..................................1111111111111111      .................................11111111111111
..................................111111..........      .................................11111.........
...................................111111111111111      ..................................1111111111111
....................................11111111111111      ...................................111111111111
....................................1111111.......      ...................................111111......
.....................................111111111N111      ....................................11111111F11
.....................................1111111111111      ....................................11111111111
......................................11N111111111      ....................................11F11111111
.......................................111111.....      .....................................111111....
.......................................11111111111      .....................................1111111111
.......................................N111111111N      ......................................11111111.
.........................................11111N111      .......................................11111F11
..........................................N1111111      .........................................111111
 ..........................................111111..     ........................................11111..
```

Figure 27. A 50x50 matrix with FP and FN error rate at 5%

```
                                                        441444444444333441331423214444424243314344224144
1111111111111111111...............................      1111111111111111111F..............................
.111N1111N11......................................      ...11111111.......................................
...1111111111.....................................      .....111111.......................................
...1N11111111N11111111............................      ......111111F1111111..............................
...11N11111111111N1111............................      .......11111111FF111..............................
.....111111111N...................................      ......11111F1.....................................
.....111111111111111111...P.......................      ......111111111111111.............................
....P..1111111N1N11...............................      ........11111F1111................................
.......1111111111.................................      ........1111111...................................
........1111111111111111..........................      .........1111111111111............................
........1111111111111N1111........................      .........1111111111F1111..........................
..........11111111111N111111......................      ..........111111F11F111111........................
..........11N111..................................      ..........1111....................................
...........1111111111111N..P.......................     ..........111111F11111............................
...........1111N111111111111......................     ............1111F111111111........................
...........1111111N11111111111....................      ..........111111FF111111111.......................
............111111111111111111....................      ...........11111F11111111111......................
............111111111N11111.......................      ...........11111F11F11111.........................
.................111111111111111N.................      ...............1F1111111111111....................
.................P11111N11N1111111................      ..................1111F11F1111111.................
..................111N1111........................      ..................11F1111.........................
..................1111111111111..................P      ..................111111111111....................
...................111111.........................      ...................11111..........................
................P...1111NN11111111................      ...................1111FF11111111F................
......................1N11N111....................      .....................1F1FFF11.....................
........................1111N1........P...........      .......................11111......................
......................P....111NN111111............      ...........................11FFF111111............
...........................111N1111111111.........      ...........................111F1111111111.........
.............................11111111111111111....      ............................1F11111111111111111...
..............................11111111N11111......      ...............................1111111F111111.....
...............................11111111...........      ...............................11111111...........
................................11111111111N1111N1      .................................11111111111F1111.
................................P1111111111111....      .................................11111111111111...
.................................1111111N111111111      ..................................1111111F11111111
.................................1111111111.......      ..................................1111111111......
..................................11N111111111111N      ...................................11F111111111111
..................................111111..........      ...................................111111.........
...................................N11111111N11111      .....................................11111111F1111
....................................11111111111N11      .....................................1111111111F11
...................P................1111111.......      .....................................1111111......
.....................................1111111111N11      ......................................111111111F11
.....................................1N1N111111111      ......................................1F1F11111111
......................................111111111111      .......................................11111111111
.......................................111111.....      ........................................111111....
..........P............................1111111N111      ........................................11111111F1
.......................................1111N111111      ........................................1111F11111
.........................................111111N1N      ..........................................11111F11
..........................................11111111      ...........................................1111111
..........................................1111N1..      ...........................................11111..
```

Figure 28. A 50x50 matrix with FP and FN error rate at 10

5.2. Results on Real Genomic Data

Another experiment we performed [Lu et al. 2001] using both STS and EST markers on HTG sequences has the following result. We did one experiment on chromosome 22 whose sequence is known. Clones were randomly generated, and the STS and EST markers were selected form STS database and UniGene database. The match of STS markers and the BAC clones are decided using a computational procedure known as "electronic PCR" (e-PCR) [GDS1997]. The match of EST markers and the BAC clones are decided using BLAST as follows. For an EST marker, if (i) the identity percentage of the highest scored local alignment of the EST and the genomic sequence is greater than 70% (80%, 90%, respectively) and (ii) the length of the above local alignment is greater half of the length of the EST sequence, then we say the EST marker matches the genomic sequence under 70% (80%, 90%, respectively) quality. Moreover, the relation of two non-overlapping clones can be determined if they share ESTs that belong to the same UniGene cluster. The experimental results are listed in Table 6. We compare the clone ordering of our program with the real ordering of the clones in chromosome 22 and count the number of contigs. The clone ordering is compared based on the positions of their left endpoints with respect to its real ordering in chromosome 22. Similar to the definition of column deviation, we can define the displacement $d(v)$ of the left endpoint, say v, of a clone in the resultant ordering. If $d(v)$ is greater than 4, we say this clone is *jump clone*. Usually, low quality matches might increase the density of markers and reduce the number of contigs, though they likely contain more errors than high quality matches. Preliminary experiment shows that our algorithm can correctly reduce an STS map from 28 islands down to 18 islands using 70% EST matches, and there are only six jump clones using 70% EST matches.

Table 6. The experimental results of chromosome 22.

EST quality	STS only	90%	80%	70%
Number of jump clones	0	4	4	6
Number of islands	28	24	24	18

In another experiment with chromosome 4 (whose sequence is unknown) we obtained the following island-reduction effect with STS and different quality of EST markers:

Table 7. The island-reduction effect of various low quality ESTs

EST quality	90%	80%	70%
# of islands	61	60	35

In our experiment, markers are ordered first and the clone ordering is determined by cluster of markers in order to construct the golden path, which is simply the way we assembled all the cosmid/bac/pac clones.

For a 400 by 400 matrix, the computation time is close to 3 minutes. Our experience indicates that, the number of FPs should be carefully controlled (in experiment) since they can severely tangle with NPs, CCs and disrupt the final ordering.

Acknowledgement

We would like to thank Professor Ueng-Cheng Yang of National Yang-Ming University for providing the EST data for our experiment. We would also like to thank the National Science Council for their generous support under Grant NSC 89-2213-E-001.

References

Alizadeh F., Karp, R.M., Newberg, L.A., and Weisser D.K. 1995. Physical mapping of chromosome: A combinatorial problem in molecular biology. *Algorithmica* 13(1/2), 52-76.

Alizadeh, F., Karp, R. M., Weisser, D. K., and G. Zweig. 1994. Physical mapping of chromosomes using unique probes. *Proc. 5th SODA*, 489-500.

Booth, K. and Lucker, G. 1976. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. of Computer and system science* 13, 335-379.

Christof, T., Jünger, M., kececioglu, J., Mutzel, P., Reinelt, G. 1997. A branch-and-cut approach to physical mapping with end-probes. *Proc. 1st Annual International Conference on Computational Molecular Biology*, 84-92.

Daues, W., J., Hanks, L., and Capra, R. 1995. Fragment collapsing and splitting while assembling high-resolution restriction maps. *J. Computational Biology* 2(2),185-205.

Eichler, E. E. 2002. Recent duplication, evolution and assembly of the Human Genome. *Proc. 6th Annual International Conference on Computational Molecular Biology*, 155.

Fulkerson, D.R. and Gross, O.A. 1965. Incidence matrices and interval graphs, Pacific Journal of Mathematics, 15 (3), 835-855.

Golumbic, M.C., Kaplan, H., and Shamir, R. 1994. On the complexity of DNA physical mapping. *Adv. in Applied Math* 15, 251-261.

Golumbic, M.C. and Shamir, R. 1993. Complexity and algorithms for reasoning about time: a graph-theoretical approach. *JACM* 40, 1108-1133.

Greenberg, D.S. and Istrail, S. 1995. Physical mapping by *STS* hybridization: Algorithmic strategies and the challenge of software evaluation. *J. Computational Biology* 2(2),185-205.

Hsu, W. L. 2002. A simple test for the consecutive ones property, *Journal of Algorithms* 42, 1-16, 2002.

Jain, M., Myers, E.W. 1997. Algorithms for Computing and integrating physical maps using unique probes. *Proc. 1st Annual International Conference on Computational* Molecular *Biology*, 151-161.

Karp, R.M.1993. Mapping the genome: some combinatorial problems arising in molecular biology. *Proc. STOC* 93, 278-285.

Lu, W. F., Hsu, W.L., Liu, N. and Yang, U.C. 2001. An Error-Tolerant Map Construction Algorithm Using EST Markers. *DNA Sequence Assembly Conference, USC*.

Mizukami, T., Chang, W.I., Garkavtsev, I., Kaplan, N., Lombardi, D., Matsumoto, T., Niwa, O., Kounosu, A., Yanagida, M., Marr, T.G. and Beach, D. 1993. A 13kb resolution cosmid map of the 14 Mb Fission Yeast Genome by nonrandom sequence-tagged site mapping. *cell* 73, 121-132.

Mott, R., Grigoriev, A., and Lehrach. H. 1994. An algorithm to detect chimeric clones and random noise in genomic mapping. *Genetics* 22,482-486.

Mayraz, G. and Shamir, R. 1999. Construction of physical maps from ologonucleotide fingerprints data. *Journal of computational biology* 6(2), 237-252.

Natanzon, A., Shamir, R. and Sharan, R. 1998. A polynomial approximation algorithm for the minimum fill-in problem. *The thirtieth annual ACM Symposium on Theory of Computing*, 41-47.

Palazzolo, M.J., Sawyer, S.A., Martin, C.H., Smoller, D.A. and Hartl, D.L. Optimized strategies for sequnce-tsagged-site selection in genome mapping. 1991. *Proc. Natl. Acad. Sci.* 88, 1991.

Yannakakis, M. 1981. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Math* 2, 77-79.