

Clustering Binary Fingerprint Vectors with Missing Values for DNA Array Data Analysis

Andres Figueroa
Dept of Comp. Sci.
Univ. of Calif., Riverside
andres@cs.ucr.edu

James Borneman
Dept of Plant Pathology
Univ. of Calif., Riverside
borneman@ucrac1.ucr.edu

Tao Jiang
Dept of Comp. Sci.
Univ. of Calif., Riverside
jiang@cs.ucr.edu

Abstract

Oligonucleotide fingerprinting is a powerful DNA array based method to characterize cDNA and ribosomal RNA gene (rDNA) libraries and has many applications including gene expression profiling and DNA clone classification. We are especially interested in the latter application. A key step in the method is the cluster analysis of fingerprint data obtained from DNA array hybridization experiments. Most of the existing approaches to clustering use (normalized) real intensity values and thus do not treat positive and negative hybridization signals equally (positive signals are much more emphasized). In this paper, we consider a discrete approach. Fingerprint data are first normalized and binarized using control DNA clones. Because there may exist unresolved (or missing) values in this binarization process, we formulate the clustering of (binary) oligonucleotide fingerprints as a combinatorial optimization problem that attempts to identify clusters and resolve the missing values in the fingerprints simultaneously. We study the computational complexity of this clustering problem and a natural parameterized version, and present an efficient greedy algorithm based on MINIMUM CLIQUE PARTITION on graphs. The algorithm takes advantage of some unique properties of the graphs considered here, which allow us to efficiently find the maximum cliques as well as some special maximal cliques. Our experimental results on simulated and real data demonstrate that the algorithm runs faster and performs better than some popular hierarchical and graph-based clustering methods. The results on real data from DNA clone classification also suggest that this discrete approach is more accurate than clustering methods based on real intensity values, in terms of separating clones that have different characteristics with respect to the given oligonucleotide probes.

Keywords: oligonucleotide fingerprinting, cluster analysis, DNA clone classification, algorithm, DNA array

1. Introduction

A DNA *array* is an orderly arrangement of DNA samples on a single chip. It provides a medium for matching DNA samples based on Watson-Crick base-pairing rules. There are various designs of DNA arrays depending on the application. In oligonucleotide fingerprinting (e.g. [9, 16, 27, 28]), a DNA array consists of thousands of spots, each of which may hold a different type of DNA sequences (also called *clones*). To perform a hybridization experiment, a drop of each type of DNA in solution is placed in a unique spot of the array and a (short) DNA sequence (i.e. an *oligonucleotide*, usually 8-50 bases), called a *probe*, is applied to hybridize with *all* the clones on the array. If the probe occurs as a substring of the clone in a spot, it will hybridize to the spot. Once the probe has been hybridized to the array and all unbound oligonucleotides have been washed off, the array is scanned to determine how much probe is bound to each spot. The experiment is then repeated for a set of probes to create *fingerprints* of the clones, where the fingerprint of a clone is simply a vector consisting of the hybridization intensity values between the clone and each probe. We observe that obtaining accurate fingerprint data can be challenging. Quantization of the intensities on each spot is subject to noise from irregular spots, dust on the chip, and nonspecific hybridization. Deciding the intensity threshold between spots and background can also be difficult, especially when the spots fade gradually around their edges (see e.g. [4]).

Oligonucleotide fingerprinting makes use of DNA arrays, and is one of the most efficient methods to characterize DNA clone libraries. It has many applications including gene expression profiling and classification of DNA clones (see, e.g. [8, 9, 13, 16, 27, 28]). In this paper, we are especially interested in the latter application, which arises in the classification of microorganisms. Here, to classify a community of microorganisms (extracted, e.g., from soil), a library of (randomly selected) ribosomal RNA genes (rDNA clones) is created. These clones are then subjected to a

series of hybridization experiments, each of which uses a single DNA oligonucleotide probe, resulting in a set of fingerprints as mentioned above. Based on a cluster analysis of these fingerprints, the rDNA clones are classified into *operational taxonomic units* (OTUs) to reflect their different characteristics with respect to the probes (*i.e.* what probes hybridized and what did not). Once classified, taxonomic descriptions of each OTU can be determined by clustering them with fingerprints of known sequences or by nucleotide sequence analyses of representative clones from the OTUs. This has already proven to be a powerful high-throughput technique for studying microorganisms [27, 28].

A crucial step in the above analysis is the clustering of clone fingerprints. Ideally, we would like to translate hybridization intensity values into binary values where 1 indicates hybridization and 0 indicates the opposite. Unfortunately, given the intensity values provided by the scanned array, it is not always easy to determine which clones hybridized and which did not due to the problems mentioned above. Hence, most of the existing methods consider real intensity values, normalize/rank them, and cluster the resulting fingerprints based on some distance/similarity measures such as Euclidean distance, Minkowsky metrics, and SIMILARITY FACTOR (SIM) for real vectors [9, 13, 19]. Although these clustering methods seem to work well in gene expression profiling, they have an obvious drawback in DNA clone classification because distance/similarity measures for real vectors usually do not treat positive and negative hybridization signals equally. Clustering results obtained using such measures tend to be more sensitive to positive signals than to negative ones.

We have recently proposed a discrete approach to the above cluster analysis in the classification of microbial rDNA clones [27, 28]. Control clones with known characteristics with respect to the probes are included in DNA array experiments to provide reference intensity values. Oligonucleotide fingerprint data are normalized and binarized using the reference values from the control DNA clones. Here, each intensity value is classified into a 1 (for hybridization) or a 0 (for no hybridization) or an N (for unknown, which is also called a *missing* value). More precisely, for each probe, let l_p denote the lowest (normalized) intensity value of any control clone that is expected to hybridize to the probe and h_n the highest intensity value of any control clone that is not expected to hybridize with the probe. Then, clones whose intensity values are greater than or equal to $\max(h_n + \epsilon, l_p)$ are given a 1 classification, where ϵ is a small constant. Clones whose intensity values are less than or equal to $\min(h_n, l_p - \epsilon)$ are given a 0 classification. All others clones are given an N classification. Then, a distance/similarity measure, *e.g.* Hamming distance (ignoring positions containing N 's), is used

in combination with some distance/similarity based clustering method, *e.g.* the well known *unweighted pair group method with arithmetic mean* (UPGMA) [24]. Although this approach treats positive and negative hybridization signals equally, it may still potentially cluster (binarized) fingerprint vectors that conflict with each other at some positions because (i) it only considers the distance between the fingerprints and (ii) the missing values are completely ignored.

In this paper, we formulate the clustering of (binarized) oligonucleotide fingerprints as a combinatorial optimization problem that attempts to identify clusters and resolve the missing values in the fingerprints simultaneously. We study the computational complexity of this formulation and its parameterized version where the maximum number of N 's in a fingerprint vector is bounded by a parameter p . Our results show that the problem and the parameterized variant (for $p \geq 3$) are NP-hard. However, the problem is polynomial-time solvable when $p = 1$. The problem can be naturally represented as finding a minimum clique partition on graphs. We present an efficient greedy algorithm based on MINIMUM CLIQUE PARTITION. The algorithm takes advantage of some unique properties of the graphs (defined from fingerprints) considered here. These properties allow us to efficiently find the maximum cliques as well as some special maximal cliques efficiently. Our experimental results on simulated and real data demonstrate that the algorithm runs faster and performs better (in the context of DNA clone classification) than popular clustering methods such as UPGMA (which was used in [27, 28]), CLUSTER [10] and CLICK [20]. The results on real data from the classification of microbial rDNA clones suggest that this discrete approach is more accurate than clustering methods based on real intensity values, in terms of separating clones that have different characteristics with respect to the given probes.

The rest of the paper is organized as follows. Before leaving this section, we include a brief review of recent work on clustering DNA array data, highlighting the ones that are most related to our approach. In Section 2, we give a mathematical formulation of the problem of clustering binary fingerprint vectors that takes into account missing values. We also study the computational complexity of the problem and a parameterized version in terms of the maximum number of missing values in a fingerprint vector. Section 3 presents the greedy clustering algorithm based on MINIMUM CLIQUE PARTITION and some implementation details (in order to make the algorithm as efficient as possible). Section 4 gives some experimental results concerning the performance of the greedy algorithm. Some conclusion remarks are provided in Section 5. Due to the page limit, all proofs are omitted in the extended abstract but can be found in the full version of the paper.

1.1. Previous Related Work on Clustering Fingerprint Data

As mentioned above, many clustering methods for DNA array data have been studied, including hierarchical methods [8, 9, 10, 22], K-means [13], greedy methods [17, 16], graph partitioning [7, 12, 20, 29], probabilistic methods [3, 5, 15, 18], and self-organizing maps [25, 26]. A recent comprehensive survey can be found in [19].

In terms of the technique involved, our new clustering algorithm is perhaps most related to the algorithms based on graph partitioning. Given a set of fingerprints and some distance/similarity measure between the fingerprints, one can define a graph whose vertices represent the fingerprints and edges are weighted with the distance/similarity values between corresponding fingerprints [7, 20, 29]. The fingerprint clustering problem then becomes the problem of partitioning the graph into a (specific) number of sub-graphs with the smallest total distance (or the largest total similarity). [12] uses a similar graph model but its edges are unweighted and two vertices are connected by an edge if their similarity value is above some fixed threshold. In some applications such as DNA clone classification, it is often difficult to decide the best distance/similarity measure to use. It could be even harder to determine the most effective threshold. Both of these decisions have strong impact on the resulting clusters and yet in practice they are usually addressed in an *ad hoc* manner [21]. In contrast, our definition of the graph only concerns the *compatibility* between binarized fingerprint vectors and thus completely avoids these hard decisions. Moreover, as we will see later, we can find the maximum cliques and some special maximal cliques (the so called *unique* maximal cliques, which contain vertices that belongs to only one maximal clique) in such compatibility graphs efficiently. This special property serves as the basis of our greedy strategy for minimum clique partition. Another advantage of our method is that we do not need to make assumptions about the intensity data like those made, for example, in [20] where the distributions of inter- and intra-cluster pairwise sample correlations are well separated and Gaussian. Xing and Karp recently demonstrated that such assumptions do not always hold [29].

As mentioned before, most approaches to cluster analysis of array data employ real intensity values. Our approach employs binary interpretations of the intensity values, with possible uncertainties. An important advantage of this discrete approach is that binarized fingerprints are essentially reproducible whereas (normalized) real intensity values are generally not. This gives another motivation for binarizing fingerprints in oligonucleotide fingerprinting. Binary (and ternary) fingerprint vectors are also used in papers such as [12, 14]. (The approach

in [12] also works for non-binary fingerprints because its clustering is based on similarity as mentioned above.) But these papers do not attempt to directly address the reliability of the deduced binary (and ternary) values, although [14] takes a conservative approach in using these values. A recent work advocating the binarization of fingerprint vectors is presented in [21]. It presents a rigorous procedure to convert real gene expression data into binary values. However, the procedure may not be suitable for applications such as DNA classification because the binarized values may not directly correspond to the occurrence of hybridization between clones and probes. Furthermore, it also does not address the reliability of the binarized values.

In [13], the reliability of hybridization intensities are evaluated using clones spotted twice. The ratio between the larger and the smaller intensity values is computed for each clone. If the ratio is above some pre-specified threshold, the intensity of the clone is considered as a missing value. However, these missing values are simply ignored in the analysis. Our approach considers missing values as part of the clustering problem, and tries to resolve them in a rigorous way. We note in passing that the idea of using control (or housekeeping) clones to help normalize and interpret DNA array data has also been used before in, *e.g.* [14].

2. The Binary Clustering Problem and Computational Complexity

As mentioned above, we consider binarized fingerprints obtained from hybridization intensity data, which are vectors of 0 (hybridization), 1 (no hybridization), or N (unknown) classifications. For convenience, call such a binarized vector a 0-1- N vector. Suppose that there are totally n clones and L probes. Then each fingerprint vector has length L . The set of all such (binarized) fingerprint vectors will be denoted by $F = \{f_1, f_2, \dots, f_n\}$, for $1 \leq i \leq n$, where f_i is a 0-1- N vector of length L corresponding to some clone. Two 0-1- N fingerprint vectors f_i and f_j are *compatible* if they do not differ at any position or at any position that they differ, say $f_i[k] \neq f_j[k]$, where $1 \leq k \leq L$, we have $f_i[k] = N$ or $f_j[k] = N$. Here, $f_i[k]$ denotes the k -th component of vector f_i . A 0-1 vector r is a *resolved vector* of a 0-1- N fingerprint vector f if $r[i] = f[i]$ for all $1 \leq i \leq L$ such that $f[i] = 1$ or $f[i] = 0$.

We will be interested in identifying clusters consisting of mutually compatible 0-1- N fingerprint vectors. Each of these clusters should potentially correspond to clones that have the same characteristics with respect to the given set of probes. Indeed, clones in such a cluster could even correspond to the same gene or genes from the same family in

microbial rDNA clone classification.¹ Following the *minimum description length* (MDL) principle (or Occam’s razor), it is natural to consider the problem of partitioning the set F into the smallest number of clusters, each consisting of mutually compatible fingerprint vectors.² Since a set of mutually compatible vectors can always be resolved in the same way, this problem is in fact equivalent to resolving the N ’s in the fingerprint vectors such that the number of distinct resulting (resolved) vectors is minimized. The problem is formally defined below.

BINARY CLUSTERING WITH MISSING VALUES (BCMV)

Instance: a set F of 0-1- N fingerprint vectors.

Feasible solution: a partition of F into disjoint subsets F_1, \dots, F_k such that for each $1 \leq i \leq k$, any two vectors in F_i are compatible.

Measure: Cardinality of the partition, to be minimized.

In practice, the number of N ’s in a binarized fingerprint vector is often upper bounded by a small constant, depending on the quality of hybridization intensity values and choice of control clones. It is thus interesting to consider the following parameterized version of BCMV, denoted as BCMV(p), where each input fingerprint vector is assumed to have at most p N ’s. Unfortunately, we can show that BCMV is NP-hard. In fact, we can prove a stronger result: BCMV(p) is NP-hard for any $p \geq 3$. The following definition will be useful.

Definition 2.1 *Given a set of 0-1- N fingerprint vectors F , define a graph $G_F = (F, E_F)$ where two vertices (fingerprints) are adjacent if and only if they are compatible. The graph G_F will be called the compatibility graph of F .*

Clearly, solving BCMV for F is equivalent to the problem of finding a minimum clique partition (MCP) on G_F . Hence, the NP-hardness of MCP [2] implies immediately the NP-hardness of BCMV. The proof can be strengthened to show the NP-hardness of BCMV(p), for any $p \geq 3$.

Theorem 2.2 *The problem BCMV(p) is NP-hard, for any $p \geq 3$.*

Interestingly, we can show that BCMV(1) has a polynomial time solution by reducing it to VERTEX COVER

¹In microbial rDNA clone classification, a gene may be represented multiple times in the clones due to random sampling. Note that, our notion of clustering differs slightly in some sense from the conventional definition that aims at grouping *similar* fingerprints according to some homogeneity measure. Our objective is to *distinguish* fingerprints that illustrate any difference, because they correspond to different genes or gene families.

²This is also consistent with the hypothesis that biomolecular diversity is a precious resource.

on bipartite graphs, which is known to have polynomial time algorithms (see *e.g.* [1]). Presently, we do not know if BCMV(2) has a polynomial time algorithm.

Theorem 2.3 *The problem BCMV(1) can be solved in polynomial time.*

Next we consider the approximability of BCMV. Since BCMV is equivalent to MCP in general and MCP is hard to approximate, it is unlikely for BCMV to have good approximation algorithms. However, we can show that BCMV(p) has constant ratio polynomial-time approximation algorithms.

Theorem 2.4 *For any p , BCMV(p) can be approximated in polynomial time with ratio 2^p .*

3. A Greedy Algorithm Based on Clique Partition

In this section, we present a greedy algorithm for BCMV(p) that runs in time $O(p2^p n^2)$. Since p is usually pretty small compared with n in practice,³ the algorithm is very efficient. We will first outline the algorithm and then show how to implement the algorithm carefully to achieve the desired efficiency.

3.1. An Outline of the Algorithm

Let F be a set of n 0-1- N fingerprint vectors of length L . Define the sets R , $R(f)$, and $F(r)$ as in the proof of Theorem 2.4, *i.e.* $R(f)$ is the set of resolved vectors of fingerprint vector f , $R = \bigcup_{f \in F} R(f)$, and $F(r) = \{f \in F \mid r \in R(f)\}$. We will consider the compatibility graph G_F and design an algorithm to partition G_F into a small set of cliques. Our algorithm is based on repeatedly finding maximal cliques in G_F . The following lemma provides an efficient way to identify maximal cliques in G_F .

Lemma 3.1 *For each maximal clique Q , there exists a resolved vector $r \in R$ such that $F(r)$ induces Q in the graph G_F .*

Note that the converse of Lemma 3.1 is not true, *i.e.* for some r , $F(r)$ may induce a clique that is not maximal. However, a set $F(r)$ with the maximum size always induces a maximum clique in G_F , and we can thus find a maximum clique in $O(2^p Ln)$ time. A pseudocode for finding a maximum clique is given in Figure 1.

Call a maximal clique in G_F *unique* if it contains at least a vertex that belongs to only one maximal clique. Clearly,

³In our microbial rDNA clone classification project, n is usually a few thousand and p is around 4, as given in Table 2.

```

Procedure FINDMAXIMUMCLIQUE ( $F$ )
  let  $r_{max}$  be the resolved vector such that
   $|F(r_{max})| = \max_{r \in R} \{|F(r)|\}$ ;
  return( $F(r_{max})$ )
end

```

Figure 1. Procedure FINDMAXIMUMCLIQUE.

it is always optimal to include unique maximal cliques in a clique partition. A straightforward algorithm for finding a unique maximal clique in a general graph might take $O(n^3)$ time, using the property that vertex v is belong only one maximal clique if and only if all its neighbors form a clique. However, by taking advantage of the special construction of G_F and the connection between resolved vectors and maximal cliques as given in Lemma 3.1, we are able to find a unique maximal clique much more quickly, as illustrated in Figure 2. The function FINDUNIQUEMAXIMALCLIQUE(F, V_i) finds a unique maximal clique in G_F and records the clique in V_i . A straightforward implementation of FINDUNIQUEMAXIMALCLIQUE would take $O(2^p n^2)$ time. In the next subsection, we will give an implementation that requires only $O(p2^p n)$ time.

```

Function FINDUNIQUEMAXIMALCLIQUE ( $F, V_i$ )
  for each  $f \in F$  do
    if  $\exists r \in R(f)$  such that  $F(t) \subseteq F(r) \forall t \in R(f)$  then
       $V_i \leftarrow F(r)$ ; return(true);
  return(false)
end

```

Figure 2. Procedure FINDUNIQUEMAXIMALCLIQUE.

Our algorithm for finding a small clique partition of G_F first looks for a unique maximal clique and removes it from the graph (and updates F accordingly). We keep finding and removing such unique maximal cliques until no unique maximal cliques can be found. Then, a greedy action takes place by removing a maximum clique from the graph. We repeat this process until all vertices of G_F have been included in some clique. A pseudocode of the algorithm, called GREEDY CLIQUE PARTITION (GCP) is given in Figure 3. We will give an implementation of GCP in the next subsection that runs in $O(p2^p n^2)$ time.

3.2. Efficient Implementation of the Algorithm

Our implementation keeps the 0-1- N fingerprint vectors $f \in F$ on a list. For each vector f , let $N(f)$ denote the

```

Algorithm: GCP ( $F, \mathcal{C}$ )
   $V \leftarrow F$ ;  $i \leftarrow 0$ ;
  repeat
    while FINDUNIQUEMAXIMALCLIQUE( $V, V_i$ ) do
       $V \leftarrow V \setminus V_i$ ;
       $V_i \leftarrow$  FINDMAXIMUMCLIQUE( $V$ );  $V \leftarrow V \setminus V_i$ ;
    until  $V = \emptyset$ ;
   $\mathcal{C} \leftarrow \{V_j \mid j = 0, 1, \dots, i\}$ ;
  return( $\mathcal{C}$ )
end

```

Figure 3. The algorithm GCP for finding a small clique partition of G_F .

positions of f that contain N values. Clearly, we can compute all $N(f)$'s in $O(nL)$ time. Each node of the list for F contains a set of addresses, one for each resolved vector $r \in R(f)$. Each address corresponds to a cell of a hash table H . Each cell of H contains four data types: (1) the size of $F(r)$, denoted by $c(r)$, (2) the set $F(r)$, (3) a vector of size L , denoted by $v(r)$, and (4) the set of the positions of N values in vector $v(r)$, denoted by $N(r)$. An open addressing with double hashing method is used for the hash table H . Under the assumption of uniformly random hashing, each hashing operation runs in time $O(1)$ on average, where the value of the constant depends on the load factor α , which is defined as the size of H divided by the number of elements stored in H . We always choose the size of H , denoted as h , such that the load factor α is at least 2. More precisely, let $m = \log_2 |R|$. We set h as the smallest prime number greater than $2^{\lceil m \rceil + 1}$.

Double hashing uses a hash function of the form $h(r, i) = (h_1(r) + i \cdot h_2(r)) \bmod h$, where r is a resolved vector, h is the size of H , and h_1 and h_2 are auxiliary hash functions, $0 \leq i \leq h - 1$. For a given resolved vector r , $h_1(r)$ is the binary number formed by $\lceil m \rceil + 1$ vector positions of r . To choose those positions, let $zero(j)$ and $one(j)$ be the total number of 0 and 1 values, respectively, in all fingerprint vectors at position $0 \leq j \leq L$. For each position j , a ratio $\gamma(j) = \frac{\min\{zero(j), one(j)\}}{\max\{zero(j), one(j)\}}$ is computed. Note that $\gamma(j) \leq 1$. We define $h_1(r)$ as the binary number formed by the $\lceil x \rceil + 1$ positions in r with the highest $\gamma(j)$ values, and, $h_2(r)$ simply as the binary number formed by all the bits in r . The hash table H is filled up as follows. For each fingerprint vector $f \in F$, the set $R(f)$ of resolved vectors is inserted into H . When a resolved vector r of some fingerprint vector f is inserted for the first time, $c(r)$ is set to one, $F(r)$ is initialized to the element f , f is copied to $v(r)$, and, $N(f)$ is copied to $N(r)$. For example, if $f = (0, 1, N, 0, 1, 1)$ and $r = (0, 1, 0, 0, 1, 1)$ is the resolved vector to be inserted into H , then $c(r) = 1$,

$F(r) = \{f\}$, $v(r) = (0, 1, N, 0, 1, 1)$, and $N(r) = \{3\}$. The next time that the same resolved vector r is inserted for another fingerprint vector g (say, $g = (0, 1, 0, N, 1, 1)$), we increment the counter $c(r)$ by one, add g to $F(r)$, copy the 0 and 1 values in g at positions $N(r)$ into the vector $v(r)$ at the same positions, and update $N(r)$ accordingly. Hence, now $c(r) = 2$, $F(r) = \{f, g\}$, $v(r) = (0, 1, 0, 0, 1, 1)$, and, $N(r) = \{3\}$. We keep repeating this process until all resolved vectors in R are inserted into H . Clearly, at most $2^p n$ insertion operation will be performed on H . Since each insertion takes at most $O(L)$ time, the total time to fill up H is bounded by $O(2^p L n)$.

To look for a maximum clique in G_F , we just need to search for the biggest number $c(r)$ in H . The corresponding set $F(r)$ will be a maximum clique in G_F . This takes $O(2^p n)$ time. To identify a unique maximal clique, we checks if any vertex f belongs to only one maximal as outlined in Figure 2 as follows. For each fingerprint vector f , we test if for all resolved vectors $r \in R(f)$, the vectors $v(r)$ restricted to positions $N(f)$ are compatible with each other. It is easy to show that f belongs to only one maximal clique if and only if all the vectors $v(r)$, $r \in R(f)$, restricted to positions $N(f)$ are mutually compatible. And if f belongs to only one maximal clique, the set $F(r)$ with the biggest $c(r)$ defines the maximal clique. Checking if a fingerprint vector f has the above property takes $O(p 2^p)$ time, since f can have at most 2^p resolved vectors. Thus, this process takes at most $O(p 2^p n)$ time totally.

In the worst case, we are able to remove only one fingerprint vector in each iteration of GCP. Hence, GCP takes $O(n)$ iterations and, the overall running time of GCP is $O(p 2^p n^2)$. In practice, much fewer of iterations are usually needed and the time for finding a unique maximal clique is bounded by $O(\bar{p} 2^p n)$, where \bar{p} denotes the average number of N 's in each fingerprint vector.

GCP has been implemented in C++ and tested on a Pentium III with a 500Mhz CPU and 512MB RAM. Figure 4(A) shows the average execution times of GCP on different parameters. For each combination of parameters, the average was taken from 10 runs. As a comparison, we have also included the running times of two popular hierarchical clustering methods: UPGMA from PAUP [24] (which was used in [27, 28]) and CLUSTER [10]. Note that, CLUSTER offers hierarchical clustering with four linkage options: centroid, single, complete, and average. We used the centroid linkage method, denoted as CLUSTERc. (CLUSTER with average linkage corresponds to UPGMA.) GCP is clearly much faster than UPGMA and slightly faster than CLUSTERc, especially when the number of N 's is small.

4. Experimental Results

We have tested the algorithm GCP on both simulated and real 0-1- N fingerprint data and compared its performance with UPGMA and CLUSTERc.⁴ We first describe the simulation test.

Define a *cluster structure* as a real vector $S = (s_1, \dots, s_d)$, where $s_i > 0$ and $\sum s_i = 1$. We say that a set of n 0-1- N fingerprint vectors has a cluster structure $S = (s_1, \dots, s_d)$ if it can be partitioned into d clusters of sizes $s_1 n, \dots, s_d n$ consisting of mutually compatible fingerprints. Our simulation program receives as input the number of 0-1- N fingerprint vectors, denoted by n , the length of the vectors, denoted by L , a cluster structure $S = (s_1, s_2, \dots, s_d)$, and a *mutation* rate $0 < \beta < 1$. First, the program generates randomly d *seed* 0-1- N fingerprint vectors of length L , with $p = \beta L$ expected N values in each vector. We make sure that no two of these vectors are compatible. For each seed vector, the N values are switched to 0 or 1 with equal probability. These resolved vectors represent the target clustering solution. We then make $s_i n - 1$ copies of the i -th resolved vector. Finally, p vector positions are randomly selected from each copy and their bits are switched to N 's. The seed fingerprint vectors and their randomly mutated copies form the input fingerprint vectors F .

We use the *Minkowski measure* (see e.g. [23]) and the *Jaccard's coefficient* (see e.g. [11]) to measure the quality of a clustering solution by comparing to target clustering. Any clustering solution for F can be represented as a binary $n \times n$ matrix A (which is in fact a submatrix of the adjacency matrix of G_F). Let T and A be the matrices for the target solution and a computed solution, respectively. Let n_{ij} be the number of entries on which T and A have values i and j , respectively. Thus, n_{11} is the number of mates that are detected by the suggested solution, n_{00} is the number of non-mates identified, while n_{10} and n_{01} count the disagreements between the true and suggested solution. The Minkowski measure is defined as $\sqrt{\frac{n_{01} + n_{10}}{n_{11} + n_{10}}}$. Hence, it measures the proportion of disagreements to the total number of mates in the target solution. A perfect solution should have score zero. The Jaccard's coefficient is defined as $\frac{n_{11}}{n_{11} + n_{10} + n_{01}}$. It represents the proportion of the correctly identified mates to the sum of the correctly identified mates plus the total number of disagreements. Hence, a perfect solution should score one.

Table 1 reports some results of the algorithm GCP on simulated 0-1- N fingerprint data. For the purpose of comparison, we have also included the results of UPGMA and

⁴We will not include comparisons with CLICK here because its current version does not allow the user to adjust the homogeneity parameter on distance data (derived from 0-1- N vectors), although the option is available for intensity data.

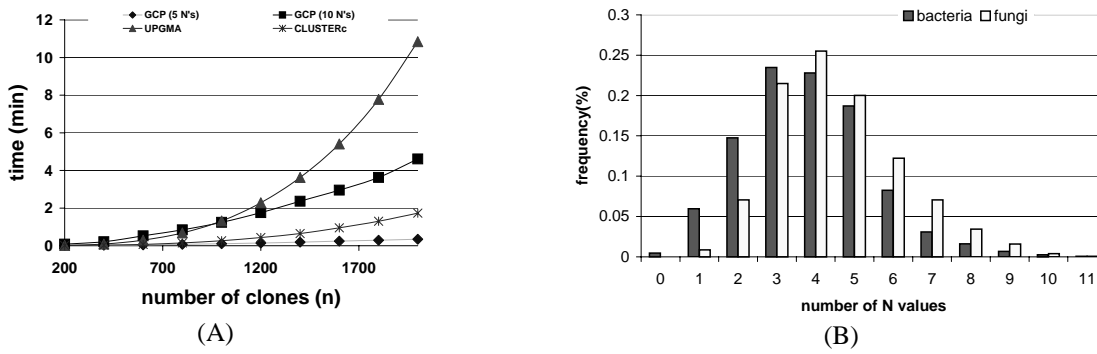


Figure 4. (A) Average execution times of GCP, UPGMA and CLUSTERc on a Pentium III in the simulations. The length of the fingerprint vectors is 25. (B) Distribution of frequencies of N 's in the binarized bacterial and fungal data sets.

n	L	d	p	GCP			UPGMA			CLUSTERc		
				Jaccard's coefficient	Minkowski measure	Number of clusters	Jaccard's coefficient	Minkowski measure	Number of clusters	Jaccard's coefficient	Minkowski measure	Number of clusters
2000	25	500	5	0.99965	0.00836	500	0.99665	0.04922	501.6	0.75497	0.49502	872.2
2000	25	500	10	0.98512	0.11789	500	0.85452	0.38725	575.6	0.35392	0.80838	1618.8
2000	25	551	5	1.0	0.0	551	0.99950	0.01976	551.8	0.85467	0.37718	892.2
2000	25	551	10	0.99388	0.07591	551	0.94449	0.23706	626.4	0.35756	0.80039	1548.2
2000	25	800	5	0.99999	0.00241	800	0.99990	0.01000	802.4	0.95228	0.21103	1176
2000	25	800	10	0.99902	0.03092	800	0.99002	0.09444	891	0.34841	0.80321	1510
2000	25	807	5	0.99944	0.01776	807	0.99847	0.03839	808.4	0.79545	0.45023	1164.8
2000	25	807	10	0.95944	0.20245	807	0.87338	0.36088	872.8	0.33675	0.81796	1643.8

Table 1. Performance of algorithms GCP, UPGMA and CLUSTERc on simulated data.

CLUSTERc on the same data sets using Hamming distance.⁵ In the test, we simulated the fingerprints with different parameters. For each set of parameters, we ran GCP, UPGMA and CLUSTERc 10 times on randomly generated data and report the average Jaccard's coefficient, Minkowski measure, and the number of clusters. GCP clearly outperformed both UPGMA and CLUSTERc on all these measures of quality, especially when the number of N 's becomes relatively large. It is worth observing that in all cases GCP was able to find the minimum number of clusters, and its solution always closely resembled the target solution (according to Jaccard's coefficient and Minkowski measure). We conjecture that GCP finds a minimum clique partition for most practical compatibility graphs G_F .

We have also compared the performance of GCP, UPGMA and CLUSTERc on two sets of real fingerprint data. The first data set is a collection of 1491 bacterial small subunit rRNA genes analyzed in [28]. The second one is a set of 1507 fungal small subunit rRNA genes studied in [27]. Two sets of 27 and 26 probes were designed based on the algorithm in [6] to hybridize the bacterial rDNA clones and

the fungal rDNA clones, respectively. The DNA array experiment for the bacterial clones used 26 control clones and the one for the fungal clones used 29 control clones. The hybridization intensities were normalized and binarized as described above. The binarized fingerprint vectors have an average of 3.84 N 's per vector in the case of bacteria and 4.54 N 's per vector in the case of fungi, as shown in Table 2 (column \bar{p}). Figure 4(B) shows the frequency distributions of N 's in the bacterial and fungal data sets.

Data set	n	L	\bar{p}	Number of Clusters		
				GCP	UPGMA	CLUSTERc
Bacteria	1491	27	3.84	769	773	991
Fungi	1507	26	4.54	556	566	870

Table 2. The numbers of clusters found by GCP, UPGMA and CLUSTERc for the bacterial and fungal data sets.

Since we do not know the true clustering solution, we could only compare the numbers of clusters (which is the object of BCMV) obtained by GCP, UPGMA and CLUSTERc. In both cases, GCP did better than UPGMA and CLUSTERc, as shown in Table 2.

⁵UPGMA as well as CLUSTERc actually produce clustering trees, but we can easily extract clusters (*i.e.* subtrees) consisting of mutually compatible fingerprints from the trees.

Since both real intensities and binarized fingerprints are available for the two real data sets, we have also compared the results of methods based on binarized values and those based on real values. We normalized and ranked the real intensity values for the bacterial and fungal data sets given in [27, 28], using the method described in [9]. We then calculated the similarity between each pair of fingerprints using SIMILARITY FACTOR (SIM) [9]. Seven methods have been compared: GCP, UPGMA and CLUSTERc on binarized fingerprints, the clustering algorithm given in [9] (denoted as DRMANAC), UPGMA on real intensities with similarity measure SIM (denoted as R-UPGMA, to avoid confusion), CLUSTERc on real intensities (denoted as R-CLUSTERc), and the CLICK on real intensities. To be fair, we adjusted the parameters (generally thresholds) in DRMANAC, R-UPGMA, R-CLUSTERc, and CLICK so that they output roughly the same number of clusters (or fewer) as GCP and their solutions were the best possible in terms of the measures considered below.⁶ Although the true clustering solutions are not known for the two real data sets, we considered the quality of the clustering solutions found by the seven algorithms in both the real domain and the binary domain. In the real domain, we measured the quality of a clustering solution as the average similarity (in terms of SIM) between fingerprint vectors contained in the same cluster [19]. In the binary domain, we considered the average number of pairs of incompatible fingerprint vectors contained in the same cluster and the average number of incompatible positions between fingerprint vectors in the same cluster. Observe that these numbers are all 0 for the solutions found by the binary methods (GCP, UPGMA and CLUSTERc). The results are summarized in Tables 3 and 4. These results show that in the real domain, DRMANAC, R-UPGMA and R-CLUSTERc did very well in terms of average similarity, but GCP, UPGMA and CLUSTERc were not too bad either. However, DRMANAC, R-UPGMA and R-CLUSTERc, surprisingly, failed considerably in the binary domain. Namely, their clusters mostly consist of fingerprints that have different characteristics with respect to the given probes. This kind of solutions would clearly not be satisfactory for applications like DNA clone classification. CLICK did poorly in the above comparisons mostly because that it was designed to only look for large clusters of sizes at least 15. Note that, for the fungal data set, the largest number of clusters output by CLICK (by trying different homogeneity values) was 113, which is much fewer than the number of clusters found by GCP.

Table 5 illustrates that the solutions found by the binary

⁶Note that, it is trivial to produce solutions with a large number of clusters that are good under these measures. However, the objective of BCMV is to minimize the number of output clusters. So, constraining all the algorithms to output roughly the same number of clusters in the following comparisons is fair to everybody.

	DRMANAC		R-UPGMA		R-CLUSTERc		CLICK	
Bacteria	0.7542	2.59	0.8064	2.75	0.7041	1.97	0.9310	4.13
Fungi	0.8315	2.21	0.6666	1.31	0.6809	1.53	0.9184	2.94

Table 4. Average incompatibility in a clustering solution. The first column is the average number of pairs of incompatible fingerprints and the 2nd column is the average number of incompatible positions between two fingerprints.

methods (GCP, UPGMA and CLUSTERc) and the real value methods (DRMANAC, R-UPGMA, R-CLUSTERc, and CLICK) on the bacterial and fungal data sets are largely different, as measured by Jaccard’s coefficient and Minkowski measure. Moreover, the solutions found by the real value methods are also significantly different from each other, compared to the difference between the solutions of the binary methods. This suggests that the existing methods for cluster analysis may not be as accurate (or mature) as people might have thought (at least as far as DNA clone classification is concerned), and perhaps more research is needed.

5. Concluding Remarks

The above experimental results clearly demonstrate that our discrete approach to cluster analysis is potentially a very effectively method for analyzing oligonucleotide fingerprints, especially in applications such as DNA clone classification. Since some hybridization intensity data may provide reliable detection of the number of occurrences of a probe in a clone up to a certain range [6], it would be interesting to extend the discrete approach to non-binary ranges $[0, t]$, where t is an integer and each missing value possibly represents a subrange of $[0, t]$. Another interesting open problem is whether BCMV(2) can be solved in polynomial time.

Acknowledgement. We would like to thank Lea Valinsky for many valuable discussions on the experimental data. The research was partially supported by a UC MEXUS/CONACYT doctoral fellowship to A.F., NSF Grant DBI-0133265 to J.B and T.J., and NSF Grants CCR-9988353 and ITR-0085910 to T.J.

References

- [1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs, N.J., 1993.

	GCP	UPGMA	CLUSTERc	DRMANAC	R-UPGMA	R-CLUSTERc	CLICK
Bacteria	769(0.5643)	773(0.5610)	991(0.6118)	749(0.8073)	751(0.8108)	794(0.6960)	639(0.4831)
Fungi	556(0.5395)	566(0.5503)	870(0.5632)	581(0.6303)	580(0.6822)	555(0.6285)	133(0.4933)

Table 3. Average similarity in a clustering solution.

	GCP		UPGMA		CLUSTERc		DRMANAC		R-UPGMA		R-CLUSTERc		CLICK	
	Jaccard	Minkowski	Jaccard	Minkowski	Jaccard	Minkowski	Jaccard	Minkowski	Jaccard	Minkowski	Jaccard	Minkowski	Jaccard	Minkowski
GCP	*	*	0.3221	1.1012	0.2430	0.9421	0.0491	3.0131	0.0407	1.8178	0.0897	1.03	0.0495	3.8879
UPGMA	0.7296	0.5788	*	*	0.2652	0.8806	0.0613	2.5934	0.0393	1.6354	0.0714	1.0189	0.0684	3.3005
CLUSTER	0.2212	0.9196	0.2011	0.9245	*	*	0.0266	0.9981	0.0319	1.0507	0.1293	1.1073	0.0246	5.7974
DRMANAC	0.0846	1.5622	0.0872	1.4927	0.0637	1.0159	*	*	0.0224	2.0716	0.0148	0.9999	0.2196	1.3242
R-UPGMA	0.0996	1.0893	0.0919	1.0762	0.1706	1.1058	0.1713	0.9356	*	*	0.0466	1.0031	0.1046	2.3941
R-CLUSTER	0.1103	0.9933	0.0983	0.9935	0.2546	1.0078	0.0705	0.9871	0.2416	0.9619	*	*	0.0131	7.5881
R-CLICK	0.0581	3.9048	0.0677	3.6318	0.0185	6.9744	0.0994	2.7936	0.0232	5.9367	0.0101	8.3262	*	*

b a c t e r i a

f
u
n
g
i

Table 5. Similarity between the clustering solutions found by GCP, UPGMA, CLUSTERc, DRMANAC, R-UPGMA, R-CLUSTERc, and CLICK.

- [2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, Berlin, 1999.
- [3] Y. Barash and N. Friedman. Context-specific bayesian clustering for gene expression data. *Proc. RECOMB 2001*, pages 12–21, 2001.
- [4] T. Beissbarth, K. Fellenberg, B. Brors, R. Arribas-Prat, J. M. Boer, N. C. Hauser, M. Scheideler, J. D. Hoheisel, G. Schütz, A. Poustka, and M. Vingron. Processing and quality control of dna array hybridization data. *Bioinformatics*, 16:1014–1022, 2000.
- [5] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *J. Computational Biology*, 6:281–297, 1999.
- [6] J. Borneman, M. Chrobak, G. Della Vedova, A. Figueroa, and T. Jiang. Probe selection algorithms with applications in analysis of microbial communities. *Bioinformatics*, 17(Suppl. 1):S39–S48, 2001.
- [7] C. Ding. Analysis of gene expression profiles: class discovery and leaf ordering. *Proc. RECOMB 2002*, pages 127–136, 2002.
- [8] R. Drmanac and S. Drmanac. cDNA screening by array hybridization. *Methods in Enzymology*, 303:165–178, 1999.
- [9] S. Drmanac, N. Stavropoulos, I. Labat, J. Vonau, B. Hauser, M. Soares, and R. Drmanac. Gene representing cDNA clusters defined by hybridization of 57,419 clones from infant brain libraries with short oligonucleotide probes. *Genomics*, 37:29–40, 1996.
- [10] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Nat'l Acad Sci USA*, 95:14863–14868, 1998.
- [11] B. Everitt. *Cluster analysis*, page 41. Edward Arnold, London, 3rd edition, 1993.
- [12] E. Hartuv, A. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, and R. Shamir. An algorithm for clustering cDNA fingerprints. *Genomics*, 66(3):249–256, 2000.
- [13] R. Herwig, A. Poustka, C. Müller, C. Bull, H. Lehrach, and J. O'Brien. Large-scale clustering of cDNA-fingerprinting data. *Genome Research*, 9(11):1093–1105, 1999.
- [14] S. Kim, E. Rougherty, Y. Chen, K. Sivakumar, P. Meltzer, J. Trent, and M. Bittner. Multivariate measurement of gene expression relationships. *Genomics*, 67:201–209, 2000.
- [15] G. McLachlan, R. Bean, and D. Peel. A mixture model-based approach to the clustering of microarray expression data. *Bioinformatics*, 18(3):413–422, 2002.
- [16] S. Meier-Ewert, J. Lange, H. Gerts, R. Herwig, A. Schmitt, J. Freund, T. Elge, R. Mott, B. Herrmann, and L. H. Comparative gene expression profiling by oligonucleotide fingerprinting. *Nucleic Acids Research*, 26(9):2216–2223, May 1 1998.
- [17] A. Milosavljević, Z. Strezosca, M. Zeremski, D. Grujić, T. Paunesku, and R. Crkvenjakov. Clone clustering by hybridization. *Genomics*, 27:83–89, 1995.

- [18] W. Pan, J. Lin, and C. Le. Model-based cluster analysis of microarray gene-expression data. *Genome Biology*, 3(2):research0009.1–0009.8, 2002.
- [19] R. Shamir and R. Sharan. Algorithmic approaches to clustering gene expression data. *Current Topics in Computational Molecular Biology* (eds. T. Jiang, Y. Xu and M. Zhang), pages 269–300, 2002.
- [20] R. Sharan and R. Shamir. Click: A clustering algorithm with applications to gene expression analysis. *Proc. ISMB 2000*, pages 307–316, 2000.
- [21] I. Shmulevich and W. Zhang. Binary analysis and optimization-based normalization of gene expression data. *Bioinformatics*, 18(4):555–565, 2002.
- [22] P. Sneath and R. Sokal. *Numerical Taxonomy*, pages 230–234. W.K. Freeman and Company, San Francisco, CA, 1973.
- [23] R. Sokal. *Clustering and classification: Background and current directions*, pages 1–15. In *Classification and clustering* Edited by J. Van Ryzin, Academic Press, 1977.
- [24] D. Swofford. *PAUP: Phylogenetic Analysis Using Parsimony version 4.0 beta 10*. Sinauer Associates, Sunderland, Massachusetts, 2002.
- [25] P. Tamayo, J. Slonim, D. Mesirov, J. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, and T. Golub. Interpreting patterns of gene expression with self-organizing maps: Methods and applications to hematopoietic differentiation. *PNAS*, 96:2907–2912, 1999.
- [26] P. Toronen, M. Kolehmainen, G. Wong, and E. Castren. Analysis of gene expression data using self-organizing maps. *FEBS Letters*, 451:142–146, 1999.
- [27] L. Valinsky, G. Della Vedova, T. Jiang, and J. Borneman. Oligonucleotide fingerprinting of ribosomal rna genes for analysis of fungal community composition. *Applied and Environmental Microbiology*, to appear, 2002.
- [28] L. Valinsky, G. Della Vedova, A. Scupham, S. Alvey, A. Figueroa, B. Yin, R. Hartin, M. Chrobak, D. Crowley, T. Jiang, and J. Borneman. Analysis of bacterial community composition by oligonucleotide fingerprinting of rna genes. *Applied and Environmental Microbiology*, 68(7):3243–3250, 2002.
- [29] E. Xing and R. Karp. Cliff: Clustering of high-dimensional microarray data via iterative feature filtering using normalized cuts. *Bioinformatics*, 17:S306–S315, 2001.