Journal of Combinatorial Optimization, 9, 199–211, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Clustering Binary Oligonucleotide Fingerprint Vectors for DNA Clone Classification Analysis

ZHIPENG CAI* zhipeng@cs.ualberta.ca
MAYSAM HEYDARI† mheydari@cs.ualberta.ca
GUOHUI LIN‡,§ ghlin@cs.ualberta.ca
Bioinformatics Research Group, Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada

Received November 22, 2004; Accepted January 23, 2005

Abstract. We considered the problem of clustering binarized oligonucleotide fingerprints that attempts to identify clusters. Oligonucleotide fingerprinting is a powerful DNA array based method to characterize cDNA and rRNA libraries and has many applications including gene expression profiling and DNA clone classification. DNA clone classification is the main application for the problem considered in this paper. Most of the existing approaches for clustering use normalized real intensity values and thus do not treat positive and negative hybridization signals equally. This is demonstrated in a series of recent publications where a discrete approach typically useful in the classification of microbial rRNA clones has been proposed. In the discrete approach, hybridization intensities are normalized and thresholds are set such that a value of 1 represents hybridization, a value of 0 represents no hybridization, and an N represents *unknown*, which is also called a missing value. A combinatorial optimization problem is then formulated attempting to cluster the fingerprints and resolve the missing values simultaneously. It has been examined that missing values cause much difficulty in clustering analysis and most clustering methods are very sensitive to them. In this paper, we turned a little back to the traditional clustering problem, which takes in no missing values but with the revised goal to stabilize the number of clusters and maintain the clustering quality. We adopted the binarizing scheme used in the discrete approach as it is shown to be typically useful for the clone classifications. We formulated such a problem into another combinatorial optimization problem. The computational complexity of this new clustering problem and its relationships to the discrete approach and the traditional clustering problem were studied. We have designed an exact algorithm for the new clustering problem, which is an A* search algorithm for finding a minimum number of clusters. The experimental results on two commonly tested real datasets demonstrated that the A* search algorithm runs fast and performs better than some popular hierarchical clustering methods, in terms of separating clones that have different characteristics with respect to the given oligonucleotide probes.

Keywords: DNA array, oligonucleotide fingerprinting, DNA clone classification, clustering, combinatorial optimization, A* search, evaluation function

1. Introduction

In a living organism, it is widely believed that thousands of its genes and their products (i.e., RNA and proteins) function in a very complicated but orchestrated way. In the past

*Supported by NSERC and CFI.
†Supported by NSERC.
‡Supported partially by NSERC, CFI, and NNSF Grant 60373012.
§To whom correspondence should be addressed.

several years, a new technology, called *DNA microarray*, has attracted tremendous interests among biologists. This technology can be used to monitor the whole genome on a single chip so that one can have a *whole* picture of the interactions among thousands of genes simultaneously. A DNA array is an orderly arrangement of known or unknown DNA samples to provide a medium for matching these samples based on Watson-Crick base-pairing rules. Various array designs exist depending on the applications. One of them is *Oligonucleotide Fingerprinting* (Drmanac et al., 1996), in which an array of oligonucleotide (20 to 80 oligos) or peptide nucleic acid (PNA) sequences (called *clones*) is synthesized either *in situ* (on-chip) or by conventional synthesis followed by on-chip immobilization. The array is then exposed to labeled sample DNA (called *probes*), hybridized, and the identity or abundance of complementary sequences is determined. Generally, a probe is a type of short, single-stranded fluorescence-labeled DNA. It will hybridize to the spot on the chip when the probe occurs as a complementary substring of the clone on the spot. After hybridization, all the unbound probes will be washed off and the hybridization intensity values between the probe and the clones can be measured. The hybridization experiment is repeated for a set of probes to create fingerprints of the clones, where a fingerprint is simply a vector consisting of the hybridization intensity values between the clone and the probes. In this way, oligonucleotide fingerprints (Drmanac et al., 1996; Tamayo et al., 1999; Valinsky et al., 2002b) are regarded as vectors containing hybridization signal intensities.

Oligonucleotide fingerprinting (Drmanac et al., 1996; Meier-Ewert et al., 1998; Tamayo et al., 1999; Valinsky et al., 2002b) is one of the best methods to characterize DNA clone libraries, which has many applications such as gene expression profiling and DNA clone classification. In particular, it offers an effective way to extensively analyze microbial communities. In this paper, we focus on the application of classification of the DNA clones, which is a problem arisen from the classifications of microorganisms (Tamayo et al., 1999; Valinsky et al., 2002b). It is observed, nonetheless, that obtaining accurate fingerprint data can be challenging. Quantification of the intensity 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 (Beissbarth et al., 2000). Nonetheless, once the fingerprints are obtained, clustering analysis can be performed on them such that the DNA clones are classified into *Operational Taxonomic Units* (OTUs) to reflect their different characteristics with respect to the probes. Furthermore, after classification, taxonomic descriptions of each OTU can be determined by clustering them with fingerprints of known sequences or by nucleotide sequence analysis of representative clones from the OTUs. This has already proven to be a powerful high-throughput technique for studying microorganisms (Valinsky et al., 2002a, 2002b).

There are many clustering methods for DNA array data studied in the literature, such as hierarchical methods (Drmanac and Drmanac, 1999; Drmanac et al., 1996; Eisen et al., 1998; Hartuv et al., 2000), K-means (Herwig et al., 1999), greedy methods (Meier-Ewert et al., 1998; Milosavljević et al., 1995), graph partitioning (Hartuv et al., 2000; Xing and Karp, 2001), probabilistic methods (Ben-Dor et al., 1999; McLachlan et al., 2002), and self-organizing maps (Tamayo et al., 1999). The readers may refer to Shmulevich and Zhang (2002) for a recent comprehensive survey. Most approaches employ real intensity values, together with a distance or similarity measurement. Often, it is difficult to decide the best distance/similarity measure to use; and it could be harder to determine the most effective distance/similarity thresholds. Obviously, these decisions have a strong impact on the resulting clusters and yet in practice they are usually addressed in an *ad hoc* manner (Shmulevich and Zhang, 2002). This reason and many others, such as binarized fingerprints are essentially reproducible whereas real intensity values are generally not, motivate the following idea of binarizing fingerprints (Hartuv et al., 2000; Herwig et al., 1999; Shmulevich and Zhang, 2002). In the discrete approaches, versus the approaches dealing with real intensity values, 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. Each intensity value is classified into a 1 (for hybridization) or a 0 (for no hybridization) or an N (for unknown). More precisely (Figueroa et al., 2003, 2004), for each probe, let l_p denote the lowest 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 to the probe. Then, clones whose intensity values are greater than $\max(h_n + \epsilon, l_p)$ are given a 1 classification, where ϵ is a small constant. Clones whose intensity values are less than $\min(h_n, l_p - \epsilon)$ are given a 0 classification. All other clones are given an N classification.

Despite a lot of research efforts have been put on the effective normalization and binarization methods, it is still in general hard to determine whether the clones are hybridized or not. In fact, most of the current methods do not offer an effective way to clearly determine whether the clones are hybridized or not. Nonetheless, once the binary ternary fingerprint vectors are obtained through the normalization and binarization methods, a distance/similarity measure such as Hamming distance can be used in combination with some distance/similarity based clustering method such as the well known *Unweighted Pair Group Method with Arithmetic Mean* (UPGMA) (Swofford, 2002). Such approach is demonstrated to treat positive and negative signals equally, but it may potentially cluster fingerprints that conflict with each other because the missing values are completely ignored. Based on such an observation, the clustering of binarized ternary fingerprints has been recently formulated into a combinatorial optimization problem (Figueroa et al., 2003, 2004), the so-called *Binary Clustering with Missing Values* (BCMV), that attempts to identify clusters and resolve the missing values in the fingerprints simultaneously. The computational complexity of BCMV and its parameterized version where the maximum number of N's in a fingerprint vector is bounded by a parameter p have been studied (Figueroa et al., 2003, 2004), as well as its experimental results on both the simulated and real data compared to some popular hierarchical clustering methods such as UPGMA (Valinsky et al., 2002a, 2002b) and CLUSTERc (Eisen et al., 1998).

Despite the fact that in most cases the output from the BCMV is considered better than the outputs from the hierarchical clustering methods, the true numbers of clusters in the test datasets are too large compared to the given numbers of fingerprint vectors and the clusters are not stable with different thresholds. In other words, the output clusters often consist of fingerprints that have different characteristics with respect to the given probes. Such a kind of solutions would clearly not be satisfactory for applications like DNA clone classification. Interpreted in another way, the thresholds that determine the fingerprint values might not be set ideal. Therefore, it is imaginably necessary that we should allow a cluster to contain vectors that disagree at some tiny amount of positions. Nonetheless, we should always keep in mind that unlike the hierarchical clustering methods that usually target the quality of a clustering solution, our goal is to find a minimum number of clusters while maintaining the clustering quality. Since missing values cause difficulty in most of the clustering methods, we adopted the scheme to set up only one intensity threshold such that there is no value N in the resultant fingerprint vectors. We then considered the clustering of these *deterministic* binary fingerprints to minimize the number of clusters.

Specifically, the input to our new combinatorial optimization problem is a given set of n binary fingerprint vectors of length L, and a parameter k which measures the extent of disagreement. The goal is to find a minimum number of clusters of the given vectors such that each cluster is represented by a *representative fingerprint vector* (RFV) and every given vector in the cluster disagrees with the RFV at at most k positions.

We studied the computational complexity of this problem (and its variant where the RFVs must be given vectors) and presented an exact algorithms for it (in Section 2). We tuned the thresholds on the real datasets from Figueroa et al. (2003, 2004) to generate many instances, and compared the performances of our proposed exact algorithm between the GCP algorithm (Figueroa et al., 2003, 2004), the hierarchical clustering algorithms UPGMA (Valinsky et al., 2002a, 2002b) and CLUSTERc (Eisen et al., 1998) (Section 3). The experimental results showed that the new discrete approach gave more stable clustering result subject to minor threshold tuning. We concluded the paper in Section 4.

2. Theoretical study: Complexities and algorithms

In the combinatorial optimization problem we are dealing with in this paper, the input is a set of n binary fingerprint vectors of length L, and a parameter k which measures the allowed extent of disagreement. The objective function is to cluster these given vectors such that each cluster can be represented by an RFV to which every given vector in the cluster disagrees at at most k positions and such that the number of clusters is minimized. For simplicity, we denote the problem as *Binary Vector Clustering with radius k* (kBVC). In a clustering result, the RFV for a cluster could be a given vector or could be a non-given vector. We denote the general version of kBVC without any constraint on the RFVs as kBVC, while denote the constrained version where the RFVs must be given vectors as *constrained* kBVC, or kCBVC for short.

We show in the next two subsections that both kBVC and kCBVC are NP-hard. We remark that between these two NP-hardness results, none of them implies the other. Both reductions in the hardness proofs are from the following NP-hard problem (Garey and Johnson, 1979), where a cubic graph is a connected graph in which every vertex has degree exactly 3.

MINIMUM VERTEX COVER ON CUBIC GRAPHS (3-VC):

INSTANCE: A CUBIC GRAPH $G = (V, E)$.

GOAL: A MINIMUM VERTEX COVER FOR G.

Given a cubic graph $G = (V, E)$, we can construct another graph H by inserting two vertices on each edge of E to produce three edges. Namely, if $(u, v) \in E$, then replace it by edges (u, u_1), (u_1, v_1), and (v_1, v). There are in total $|V| + 2|E|$ vertices and $3|E|$ edges in the new graph H, which is no longer cubic. Clearly, there is no triangle, or length-3 cycle, in graph H and the maximum degree of vertices in H, denoted as $\Delta(H)$, is 3. Also it is trivial to check that there is a vertex cover of size p in G if and only if there is a vertex cover of size $p + |E|$ in H.

2.1. *kBVC is NP-hard*

In the hardness proof of kBVC, the reduction starts from the graph H constructed from a cubic graph G as in the above. For ease of presentation, we denote the vertex set of graph H as V and the edge set as E, i.e., $H = (V, E)$.

Theorem 2.1. *kBVC is NP-hard.*

Proof: Given the graph $H = (V, E)$ constructed in the above way, we know that there is no triangle in it and $\Delta(H) = 3$. Assume that $|E| = n$ and $|V| = m$. Label the vertices in V as $1, 2, \ldots, m$. For every edge $(i, j) \in E$, we construct a binary vector of length km such that the entries $\ell m + i$ and $\ell m + j$ contain 1, for $\ell = 0, 1, \ldots, k - 1$, and all the other entries contain 0. This gives us an instance of kBVC containing n binary vectors of length $L = km$. In the following, we will prove that graph H has a vertex cover of size p if and only if the kBVC instance has a clustering containing p clusters. For simplicity, we assume from now on $k = 1$. The readers should be able to check that the arguments hold for any $k \geq 1$.

The "only if" part is easy. Given a vertex cover C for graph H, say $C = \{i_1, i_2, \ldots, i_p\}$, we let the vectors, each of which contains only one 1-entry (the position of the 1-entry is i_q), be the RFVs. It follows that the vectors corresponding to the edges covered by vertex i_q can be included into the cluster with its RFV being the qth RFV, where every vector differs from its RFV by exactly one position. Since all edges are covered by the vertices in C, these p clusters form a clustering.

For the "if" part, we assume that there is a clustering containing p clusters. For every cluster, we claim that it couldn't contain two given vectors corresponding to two non-adjacent edges, for the reason that these two non-adjacent vectors disagree at exactly 4 positions and thus cannot be represented by a common RFV (note that $k = 1$). In other words, vectors/edges belonging to a cluster must be pair-wise adjacent to each other. Nonetheless, recall that there is no triangle in graph H. We further claim that vectors/edges in a cluster all incident at a common vertex. Saying in the other way, these vectors all have 1 in certain position. Let the RFV be the vector which has 1 at the same "certain" position and 0 in all the other positions. It certainly disagrees with each vector at exactly one position. Therefore, we conclude that for every cluster we can determine its RFV in linear time. Let C denote the subset of vertices each of which labels the position where some RFV has a value 1. Clearly C contains at most p vertices and it is a vertex cover for graph H. This proves the theorem. □

2.2. *kCBVC is NP-hard*

In the hardness proof of kCBVC, the reduction again starts from graph $H = (V, E)$. For simplicity, we only prove the NP-hardness for 1CBVC. The proof for the general kCBVC can be similarly done.

Theorem 2.2. 1*CBVC is NP-hard.*

Proof: Given the graph $H = (V, E)$ constructed in the above way, we know that there is no triangle in it and $\Delta(H) = 3$. Assume that $|E| = n$ and $|V| = m$. Label the vertices in V as $1, 2, \dots, m$. For every edge $(i, j) \in E$, we construct a binary vector of length m such that two entries i and j contain 1 and all the other $(m - 2)$ entries contain 0; Such a vector is an *edge vector*. For every vertex $i \in V$, we construct a binary vector of length m such that only the ith entry contains 1 and all the other $(m - 1)$ entries contain 0; Such a vector is a *vertex vector*. Besides these $n + m$ vectors, there is a *zero vector* which is of length m too and all entries contain 0. This gives us an instance of 1CBVC containing $n + m + 1$ binary vectors of length m. In the following, we will prove that graph H has a vertex cover of size p if and only if the 1CBVC instance has a clustering containing $p + 1$ clusters.

The "only if" part is again easy. Given a vertex cover C for graph H, say $C = \{i_1, i_2, \dots, i_p\}$, we let the vertex vectors corresponding to the vertices in C and the zero vector be the RFVs. It follows that all edge vectors can be included into those p clusters with their RFVs being vertex vectors, and all vertex vectors corresponding to vertices outside of C can be included into the cluster with the zero vector RFV.

For the "if" part, we assume that there is a clustering containing $p + 1$ clusters. We assume further that $p < n - 1$ for otherwise every subset of $(n - 1)$ vertices can be set to be a vertex cover. In what follows, we will show that if there is an RFV which is an edge vector, then we may replace it by some vertex vector with the total number of RFVs remained the same or decreased by 1. For this purpose, we do not distinguish edge and edge vector, neither vertex and vertex vector. Suppose without loss of generality that $(1, 2) \in E$ is an RFV. For the same reason as in the proof of Theorem 2.1, the cluster that $(1, 2)$ represents, denoted as U, should not contain any other non-adjacent edges. If U contains at most one of the vertices 1 and 2, then we may simply use the vertex vector contained in U (or vertex vector 1 if U contains no vertex vectors) to be a new RFV to represent U. Therefore, we may assume that $U = \{1, 2, (1, 2)\}$. Similarly, we may assume that none of 1, 2, or zero vector is an RFV. For the non-trivial case, we must have $p \geq \frac{n}{3} > 2$. Suppose $(1, 3)$ is another edge incident at vertex 1 (note that vertex 1 has degree at least 2). There are two possible cases: (1) $(1, 3)$ is an RFV; (2) 3 is an RFV.

In case (1), we can replace RFVs $(1, 2)$ and $(1, 3)$ by two new RFVs 1 and the zero vector. In case (2), since we know that there is no edge connecting 2 and 3, we conclude that there is some edge other than $(1, 3)$ incident at 3. Assume without loss of generality that $(3, 4)$ is in E. Let U' denote the cluster represented by 3. Clearly, vector 4 cannot be in U' and thus it must be represented by some other RFV x which is either itself or some edge vector incident at 4. If there is a third edge incident at vertex 3, say $(3, 5)$, we do the same as we did for edge $(3, 4)$ (and assume the RFV is y). In either case, we replace RFVs $\{(1, 2), 3, x\}$ by RFVs $\{1, 4, \text{zero vector}\}$ (or replace RFVs $\{(1, 2), 3, x, y\}$ by RFVs $\{1, 4, 5, \text{zero vector}\}$).

Thus far, we have proven that every RFV must be either a vertex vector or the zero vector. Note that $p + 1 < n$ and that two vertex vectors disagree at exactly two positions. We conclude that zero vector must be an RFV. Furthermore, zero vector cannot represent any edge vector and thus all edge vectors are in those p clusters except that the cluster with zero vector RFV. The subset of vertices, each of which is an RFV, is a vector cover for graph H. This finishes the proof. □

2.3. *An exact algorithm*

These two NP-hard problems, kBVC and kCBVC, have some seemingly nice structures. But we so far do not have any good approximation algorithms for them, except by reducing them to the Set Cover problem. Notice that since every cluster contains at most $\sum_{i=0}^{k} \binom{L}{i} \leq 2L^k$ distinct vectors, the problems can be approximated within a factor of $k \log L$.

Theorem 2.3. *Both the kBVC and the kCBVC problems can be approximated within a factor of $O(k \log L)$, where L is the length of the binary vectors.*

The rest of this section is devoted to an exact algorithm for the 1BVC problem, which is based on a heuristic search algorithm, A* search, developed originally in Artificial Intelligence. To achieve better performance, we proposed several carefully designed heuristics and evaluation functions to speed up the search. The algorithm can be extended to kBVC for any $k > 1$, and kCBVC as well.

A* is a Best-First search algorithm which has been extensively used in many areas of Artificial Intelligence, and has been successfully applied to various bioinformatics problems, the most notable of which is probably *Multiple Sequence Alignment* (Takahiro and Hiroshi, 1994). For each state (or a node in the search tree produced by the algorithm), A* uses both the exact distance from the root state (in 1BVC, it refers to the number of clusters found so far), which is denoted as g, and a heuristic estimate of the remaining distance to the goal state (in 1BVC, it refers to the number of expected new clusters to be formed), which is denoted as h. The state with the smallest $(g + h)$ value is always expanded next by the algorithm and the first solution found by the algorithm is guaranteed to be optimal (i.e. a minimum number of clusters) provided that h always underestimates the true distance to the goal state.

Let the given vectors be $v_1, v_2, \ldots, v_n$, each is of length L. The search algorithm starts with finding all the possible RFVs that need to be considered, say $r_1, r_2, \ldots, r_m$, and for each possible RFV r_j, finding its cluster of represented given vectors, rg_j. In more details, the algorithm checks for every pair of given vectors (v_{i_1}, v_{i_2}) to determine the possible RFVs that can represent them. If one vector cannot be co-represented with another vector by some RFV, then itself forms a singleton cluster and is subsequently removed from further consideration. At the time all the RFVs are found through the above way, the associated clusters are also identified (each of which contains at least 2 given vectors).

In the second step, the algorithm sorts the RFVs in the order of non-decreasing size of their associated clusters. Assume without loss of generality that they are sorted into an order $r_1, r_2, \ldots, r_m$. Assume also that none of the associated clusters is a subset of another, for otherwise it is removed from further consideration. If there is any given vector which belongs to only one cluster, then the cluster must sit in the final solution and it is subsequently removed from further consideration. At the end of the second step, the clusters are mutually exclusive and every given vector either belongs to none of them or belongs to at least 2 of them.

In the third step, the search formally starts. At every state in the search tree, the algorithm picks one maximum-size cluster, say rg_x, from the remaining list of clusters. One arbitrary vector in rg_x, say v_x, is chosen by the algorithm to expand the search. Note that v_x belongs to at least 2 clusters (including rg_x). Therefore, there are at least 2 different ways to cluster v_x. The child states of the current state one-to-one correspond to the different ways of clustering v_x. Namely, in every child state, one cluster containing v_x is appended into the partial solution, and the given vectors in the cluster are marked as *clustered* and are subsequently removed from further consideration (along the branch). At each child state, heuristics are applied to remove the produced singleton clusters and the given vectors belonging to a unique cluster (with possible inclusion of the clusters to the partial solution). The remaining configuration is the same as that at its parent state where the clusters are mutually exclusive and every given vector either belongs to none of them or belongs to at least 2 of them. At this point, both g and h are updated accordingly. The algorithm proceeds the search until the first complete solution is found, i.e. the current state is a leaf state. A high-level pseudo-code description of the A* exact algorithm is depicted in Figure 1. For convenience, the process to remove singleton clusters is referred to as *heuristic* #1, and the process to remove the given vectors belonging to a unique cluster is referred to as *heuristic* #2.

The A* search algorithm is a general algorithmic framework that can be used to solve many hard optimization problems. The performance of the algorithm relies on a number of factors, among which the most important ones are the heuristic evaluation functions that estimate the h value. It is desirable that the evaluation functions run fast and give accurate estimation. In practice, however, these two objectives have to compromise each other. In our case, the evaluation function is designed to estimate how many clusters are necessary to include all the remaining given vectors. The problem is reduced to finding a maximum clique in a graph, as in the follows: For each state, the heuristics constructs a graph $G = (V, E)$, where V contains all the remaining given vectors and a pair of vectors

```
function A_Star($state$)
    if Solution_found( )
        return $f(state)$
    for each successor $u_i$ of $state$ do
        apply $heuristic$ #1
        apply $heuristic$ #2
        $f(u_i) = g(u_i) + h(u_i)$
        add $u_i$ to List $T$
    remove $state$ to List $T$
    find a state $new\_state$ in List $T$ having the minimum $f$ value
    A_Star($new\_state$)
```

Figure 1. An A* implementation for 1BVC. Function Solution_found( ) checks if a solution has been found; Function $f()$ returns the exact distance from root state to state u_i; Function $h()$ is the heuristic evaluation function. Our implementation also uses an open list T to store the states waiting to be expanded.

are adjacent if and only if they cannot belong to a common cluster. It then uses a greedy algorithm to find a clique to include as many vertices as possible. It is easy to see that the number of vertices in the clique serves as a lower bound on the number of clusters that are necessary to cluster all the remaining given vectors. The heuristics returns this number to h. The greedy algorithm for finding a clique sorts the vertices into a non-decreasing degree order and uses the order to iteratively include the vertices to form a clique. In Figure 1, $h(\,)$ denotes the above heuristics evaluation function.

3. Experimental results

We tested our A* search algorithm on two real fingerprint datasets, courteous to the authors of Figueroa et al. (2003). One of them is the bacterial small subunit rRNA genes where the number of vectors is $n = 1491$ and the length of the vectors is $L = 27$; the other is the fungal small subunit rRNA genes where $n = 1507$ and $L = 26$.

3.1. Stability analysis for the BVC model

The first experiment is designed to validate the stability of the BVC model for fingerprint clustering. We set different thresholds, which are close to the best thresholds determined previously, to each of the two real datasets to obtain a number of instances. As we mentioned in the Introduction, if a real intensity value is larger than or equal to the threshold, it is set to 1; otherwise, it is set to 0. The A* search algorithm was applied on every instance to find a minimum number of clusters (see Table 2). Two plots in Figure 2 show that

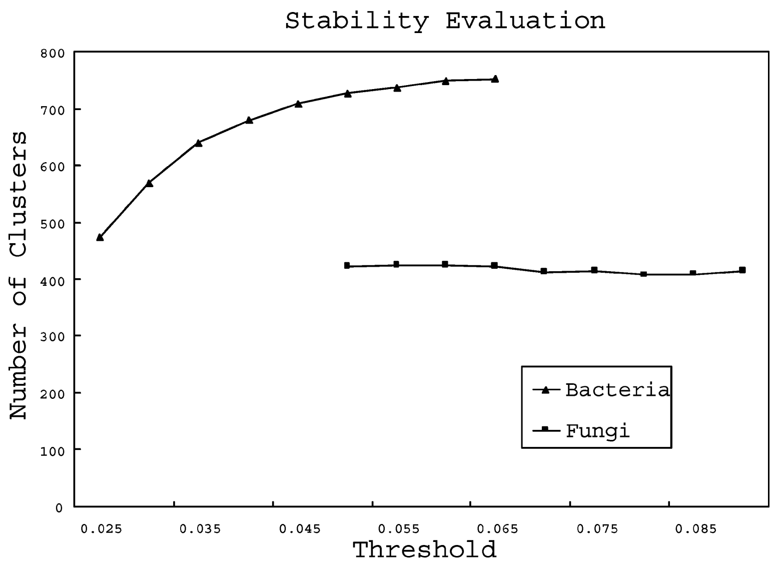

Figure 2. The number of clusters returned by 1BVC on two real datasets.

the lines are quite smooth, especially for the second dataset. This indicates that a certain error in threshold in our BVC model for fingerprint clustering is allowed and the model outputs a number of clusters, which is always close to the true number of clusters. Since the threshold determination is very hard in general, in the discrete approach in Figueroa et al. (2003), two values are set which represents a lower bound and an upper bound on the threshold, respectively. We agree that this is a way to conquer the problem but the overhead is to determine two values, although their determination becomes a bit easier. In our BVC model, however, only one threshold need to be set, and as long as it is close to the true value, the number of clusters can be roughly detected. We remark that this could be an advantageous aspect for the BVC model.

In Table 1, we compared the qualities of two groups of clustering results associated to two different threshold values for both datasets. For $threshold_1$, we selected 100 largest clusters in the clustering result to compare to the clustering result for $threshold_2$. The "common center" column records the number of RFVs for the 100 largest clusters that are remained in the clustering result for $threshold_2$. The next column records the sum of the hamming distances of the RFVs for the 100 largest clusters to the RFVs for the 100 largest clusters in the result for $threshold_2$. It should be noted that the entry divided by 100 gives the average pairwise distance. The next column records the similarity between these two clustering results, again using the 100 largest clusters, $\frac{1}{100}\sum_{i=1}^{100}\frac{|rg_i^1|}{|rg_i^2|}$, where rg_i^j is the ith largest cluster for $threshold_j$ for $j = 1, 2$. The "shift RFVs" column is the average difference of the RFVs of the 100 clusters for $threshold_1$ and $threshold_2$. Table 1 clearly shows that

Table 1. Comparison between two clustering results for two different threshold values.

Dataset	*Threshold*$_1$	*Threshold*$_2$	Common center	Hamming distance of RFVs	Similarity (%)	Shift RFVs (%)
Bacteria	0.025	0.030	27	84	30.40	3.1
	0.030	0.035	32	85	34.76	3.1
	0.035	0.040	26	92	31.77	3.4
	0.040	0.045	33	80	38.71	3.0
	0.045	0.050	32	82	42.85	3.0
	0.050	0.055	36	77	44.65	2.9
	0.055	0.060	36	83	47.13	3.1
	0.060	0.065	42	73	51.34	2.7
Fungi	0.050	0.055	47	57	50.29	2.2
	0.055	0.060	55	50	55.23	1.9
	0.060	0.065	65	46	62.20	1.8
	0.065	0.070	69	35	65.22	1.3
	0.070	0.075	69	35	68.38	1.3
	0.075	0.080	71	33	70.15	1.3
	0.080	0.085	77	25	73.22	1.0
	0.085	0.090	63	40	63.82	1.5

the clusters obtained by BVC model for different threshold values are very similar to each other. Together with Figure 2, Table 1 shows that BVC model can stand the threshold error quite well.

3.2. The comparison to other clustering methods

The experiment has also been designed to make comparison of the BVC results (returned by the A* search algorithm) to the clustering results of UPGMA and CLUSTERc. Similarly as in Figueroa et al. (2003), we used the Minkowski measure and Jaccard's measure to compare the difference of a clustering solution by comparing it to target clustering result (which is the BVC result). To do this, let a binary $n \times n$ matrix A represent the BVC clustering results, where $A_{ij} = 1$ means vectors v_i and v_j are in one common cluster. Similarly, let B represent the clustering result returned by another algorithm. Call A the target solution and B the suggested solution. Set n_{ij} as the number of entries where A and B have value i and j, respectively. Therefore, n_{11} records the number of mates that are same as the suggested solution, n_{00} is the number of non-mates position. n_{10} and n_{01} are the disagreements between the suggested solution and target solution. The Minkowski

Table 2. The clustering results returned by UPGMA and CLUSTERc on two real datasets compared to the 1BVC clustering results returned by the A* search algorithm.

Dataset	Threshold	1BVC	UPGMA	Minkowski coefficient of UPGMA	Jaccard's coefficient of UPGMA	CLUSTERc	Minkowski coefficient of CLUSTERc	Jaccard's coefficient of CLUSTERc
Bacteria	0.025	474	878	0.9334	0.344	708	1.09	0.274
	0.030	569	796	1.17	0.229	756	0.978	0.303
	0.035	641	1002	0.941	0.295	698	0.866	0.29
	0.040	679	730	1.2	0.27	753	0.987	0.31
	0.045	708	737	1.2	0.254	771	1.032	0.287
	0.050	727	743	1.2	0.25	765	1.231	0.309
	0.055	738	717	1.2	0.27	736	1.1	0.311
	0.060	750	703	1.2	0.29	754	1.08	0.331
	0.065	752	679	1.3	0.29	765	1.129	0.28
Fungi	0.050	423	634	1.2	0.24	643	1.1	0.28
	0.055	426	627	1.2	0.25	653	1.09	0.301
	0.060	425	682	1.1	0.31	678	0.981	0.26
	0.065	423	694	1.1	0.32	669	1.1	0.29
	0.070	413	727	0.96	0.35	647	1.3	0.35
	0.075	416	687	1.2	0.22	674	1.2	0.29
	0.080	409	687	1.0	0.34	646	1.1	0.30
	0.085	411	687	0.96	0.39	662	1.0	0.31
	0.090	415	707	0.91	0.39	685	1.0	0.31

measure is calculated as $\sqrt{\frac{n_{01}+n_{10}}{n_{11}+n_{10}}}$ (Figueroa et al., 2003). Intuitively, if these two solutions are similar to each other, the Minkowski measure should be close to zero. The Jaccard's coefficient is defined as $\frac{n_{11}}{n_{11}+n_{10}+n_{01}}$ (Figueroa et al., 2003), which should be close to 1 if these two solutions are similar. Table 2 summaries the comparison results between the BVC clustering results and the clustering results returned by UPGMA and CLUSTERc. From the statistics, we found that the clustering results by UPGMA and CLUSTERc do not map well to the 1BVC clustering results. There are a number of reasons behind this, one of them is that it is generally difficult to set a cut value in these two clustering methods to report a meaningful clustering.

4. Conclusions and future work

We proposed a new discrete approach for clustering binary fingerprint vectors, especially for applications such as DNA clone classification. The experimental results on two commonly used real datasets demonstrated that this model returned a stable number of clusters under a minor change in the intensity threshold. The clustering results were also stable. The comparison made to the clustering results by UPGMA and CLUSTERc suggested that some classical hierarchical clustering methods might not be immediately applicable to binary fingerprint data. Nonetheless, we do not intend to claim that our model is superior to existing models, but rather than that, our model could return more stable clustering results when the intensity threshold value can not be to determined precisely. The A* search algorithm we developed runs fast and guarantees the optimality. It might be potentially useful to bench mark real datasets for future clustering model/method development. Through some efforts, we didn't come up with better approximation algorithms for either kBVC or kCBVC, than the ratio stated in Theorem 2.3. It is an interesting question to ask if kBVC and kCBVC do possess good structures for better approximation.

References

T. Beissbarth, K. Fellenberg, B. Brors, R. Arribas-Prat, J.M. Boer, N.C. Hauser, M. Scheideler, J.D. Hoheisel, G. SchSutz, A. Poustka, and M. Vingron, "Processing and quality control of DNA array hybridization data," *Bioinformatics*, vol. 16, pp. 1014–1022, 2000.

A. Ben-Dor, R. Shamir, and Z. Yakhini, "Clustering gene expression patterns," *Journal of Computational Biology*, vol. 6, pp. 281–297, 1999.

R. Drmanac and S. Drmanac, "cDNA screening by array hybridization," *Methods in Enzymology*, vol. 303, pp. 165–178, 1999.

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*, vol. 37, pp. 29–40, 1996.

M. Eisen, P. Spellman, P. Brown, and D. Botstein, "Cluster analysis and display of genome-wide expression patterns," in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 95, pp. 14863–14868, 1998.

A. Figueroa, J. Borneman, and T. Jiang, "Clustering binary fingerprint vectors with missing values for DNA array data analysis," in *Proceedings of the Second IEEE Computer Society Computational Systems Bioinformatics Conference (CSB'03)*, 2003, pp. 38–47.

A. Figueroa, J. Borneman, and T. Jiang, "Clustering binary fingerprint vectors with missing values for DNA array data analysis," *Journal of Computational Biology*, vol. 11, pp. 887–910, 2004.

M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.H. Freeman and Company, San Francisco, 1979.

E. Hartuv, A. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, and R. Shamir, "An algorithm for clustering cDNA fingerprints," *Genomics*, vol. 66, pp. 249–256, 2000.

R. Herwig, A. Poustka, C. MiSuller, C. Bull, H. Lehrach, and J. O'Brien, "Large-scale clustering of cDNA-fingerprinting data," *Genome Research*, vol. 9, pp. 1093–1105, 1999.

G. McLachlan, R. Bean, and D. Peel, "A mixture model-based approach to the clustering of microarray expression data," *Bioinformatics*, vol. 18, pp. 413–422, 2002.

S. Meier-Ewert, J. Lange, H. Gerts, R. Herwig, A. Schmitt, J. Freund, T. Elge, R. Mott, B. Herrmann, and H. Lehrach, "Comparative gene expression profiling by oligonucleotide fingerprinting," *Nucleic Acids Research*, vol. 26, pp. 2216–2223, 1998.

A. Milosavljević, Z. Strezosca, M. Zeremski, D. Grujić, T. Paunesku, and R. Crkvenjakov, "Clone clustering by hybridization," *Genomics*, vol. 27, pp. 83–89, 1995.

I. Shmulevich and W. Zhang, "Binary analysis and optimization-based normalization of gene expression data," *Bioinformatics*, vol. 18, pp. 555–565, 2002.

D.L. Swofford, "Paup: Phylogenetic analysis using parsimony, 2002," Version 4.0 Beta 10.

I. Takahiro and I. Hiroshi, "Fast A* algorithms for multiple sequence alignment," in *Genome Informatics Workshop 94*, 1994, pp. 90–99.

P. Tamayo, J. Slonim, D. Mesirov, J. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, and T. Golub, "Interpreting patterns of gene expression with selforganizing maps: methods and applications to hematopoietic differention," in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 96, pp. 2907–2912, 1999.

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*, vol. 68, pp. 5999–6004, 2002a.

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 rRNA genes," *Applied and Environmental Microbiology*, vol. 68, pp. 3243–3250, 2002b.

E. Xing and R. Karp, "Cliff: Clustering of highdimensional microarray data via iterative feature filtering using normalized cuts," *Bioinformatics*, vol. 17, pp. S306–S315, 2001.