TCLUST: A Fast Method for Clustering Genome-Scale Expression Data

Banu Dost, Chunlei Wu, Andrew Su, and Vineet Bafna

Abstract—Genes with a common function are often hypothesized to have correlated expression levels in mRNA expression data, motivating the development of clustering algorithms for gene expression data sets. We observe that existing approaches do not scale well for large data sets, and indeed did not converge for the data set considered here. We present a novel clustering method TCLUST that exploits coconnectedness to efficiently cluster large, sparse expression data. We compare our approach with two existing clustering methods CAST and K-means which have been previously applied to clustering of gene-expression data with good performance results. Using a number of metrics, TCLUST is shown to be superior to or at least competitive with the other methods, while being much faster. We have applied this clustering algorithm to a genome-scale gene-expression data set and used gene set enrichment analysis to discover highly significant biological clusters. (Source code for TCLUST is downloadable at http://www.cse.ucsd.edu/~bdost/tclust.)

Index Terms—Microarray expression, clustering, graph algorithms, coconnectedness.

1 Introduction

With the current mRNA expression profiling technology, expression levels of tens of thousands of genes across hundreds of conditions are measured simultaneously. Genes with a common function are often hypothesized to have correlated expression levels across different conditions. It is not surprising that clustering algorithms have been intensively studied for analyzing gene expression data in order to detect the groups of genes with correlated expression patterns. However, current clustering algorithms commonly used in the domain of gene expression do not scale well for genome-scale expression data.

In the context of gene expression data, it is convenient to think of clustering on a graph in which the vertices correspond to genes (or, probe sets), and edge weights reflect the similarity/correlation between the expression profiles of the probe sets. Generally, the correlation graph is a complete graph with $n \sim 10^5$ nodes, and $m \sim 10^9$ edges. It is a common strategy to sparsify the graph by discarding the edges with edge weights smaller than a chosen threshold. In our experiments, we observe that even after filtering with a reasonable threshold, the number of edges remains $m \sim 10^7$, resulting in a graph of complexity $mn \sim 10^{11}$. Therefore, it is critical to devise a clustering algorithm that will scale well with very large graphs.

In this paper, we propose a fast method to identify the dense subgraphs in correlation graph defined on genes (or, probe sets). If clusters of functionally related genes were all coexpressed, and those were the only coexpressed genes, the correlation graph should look like a collection of disjoint cliques. Errors and other biological variations will add additional edges and remove some true edges. To simplify exposition, we will consider these extra and missing edges as erroneous data (false positives (FP), and negatives (FN), respectively), even though they might be encoding a true biological phenomenon. The resulting graph is therefore a sparse graph with "dense subgraphs" embodying functionally related clusters.

Assuming the graph has an underlying clique structure, identification of such dense subgraphs is known as "cluster editing problem" in literature. The problem has been proven to be NP-hard for arbitrary FP rates [1], [2], even when FN=0. (the clique problem; [3].) In practice it is hard for even moderate error rates. Fortunately, in the specific domain of gene expression, the error rates appear to be low enough for the approach to be viable.

Among the more popular algorithms for clustering gene expression data are K-means, SOM, hierarchical clustering, and CAST [4], [5], [6]. Previous studies comparing traditional clustering methods in microarray data have shown that K-means and SOM have superior performance to hierarchical clustering [7]. CAST has also been applied on expression data with good results [6]. However, in our experiments, they do not appear to scale well for large expression data sets, as discussed later in the text. Both K-means and CAST are similar in one sense as they both dynamically update clusters by assigning and removing vertices iteratively, according to a stated objective. In K-means, the objective is to reduce the intracluster variation, while increasing the intercluster variation. However, the number of clusters (K) is an important parameter, and must be specified in advance.

CAST updates clusters with no prior knowledge of the number and size of the clusters. It constructs one cluster at a time by iteratively examining each vertex relationship to an open, nonstabilized, cluster. CAST then uses affinity of a

- *B. Dost and V. Bafna are with the Department of Computer Science and Engineering, University of California, San Diego, CA, 92093. E-mail: {bdost, vbafna}@cs.ucsd.edu.*
- *C. Wu and A. Su are with The Genomics Institute of the Novartis Research Foundation, 10675 John J. Hopkins Drive, San Diego, CA 92121. E-mail: {cwu, asu}@gnf.org.*

Manuscript received 18 Nov. 2008; revised 22 June 2009; accepted 28 Nov. 2009; published online 30 Apr. 2010.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2008-11-0201. Digital Object Identifier no. 10.1109/TCBB.2010.34.

vertex to the cluster to determine whether the vertex belongs or not. Affinity of a vertex v to an open cluster C is measured by the sum of the edge weights going from v to the current members of C. CAST alternates between adding and removing vertices until all members but not nonmembers of C have high affinity to C. The key parameter here is *affinity* which determines the number and sizes of the clusters. For very large data sets, it takes a lot of time as it will take many iterations for the vertex $\leftrightarrow$ cluster relationship to stabilize.

The weighted cluster editing problem (defined below) has also been directly studied by Rahmann et al. [8]. The authors define a cost function, and describe a fixed-parameter algorithm which reaches its limit above 50 vertices. They also suggest an alternative fast, $O(n^2)$, layout-based heuristic for large graphs. While faster, it is still computationally intensive. In self-reported results, the time increased exponentially, requiring $\sim 10^5$ s on graphs of complexity of 10^8. Additionally, it requires setting of up to five input parameters.

We observe that existing approaches do not scale well for large data sets, and indeed did not converge for the data set considered here. In our experiments, we use microarray data acquired from 11 tissues in each of 29 inbred strains of mice Affymetrix MOE430v2 GeneChips (45,101 probe sets). This data set represents a "genetical genomics" or "eQTL" microarray experiment [9]. While the vertex set is large, it is sparse in the edges. Of the $\sim 10^9$ possible edges, 4.2×10^7 exceed a correlation coefficient of 0.3, resulting in graphs of complexity 10^{11}.

2 Weighted Cluster Editing Problem

Unweighted cluster editing problem is to make the fewest number of changes to the edge set of an input graph such that the resulting graph is a disjoint union of cliques. In this paper, we address the "weighted cluster editing problem" in the context of gene expression data which also takes the edge weights into account.

2.1 Definitions and Notation

Following Rahmann et al. [8], we consider a set of genes V and a symmetric function $s : V^2 \rightarrow R^+$ that reflects the similarity between the expression profiles of the genes in V. Given a weighted undirected correlation graph $G = (V, E)$, where $E = \{(u, v) | s(u, v) > 0\}$, our goal is to edit G by removing and adding edges in such a way that it becomes a union of disjoint cliques where each clique corresponds to subsets of genes with highly correlated expression profiles.

Each operation incurs a nonnegative cost: If $(u, v) \in E$, the edge removal cost of (u,v) is $c^- = s(u, v)$. If $(u, v) \notin E$, the edge addition cost of (u,v) is $c^+ = -s(u, v)$. Note that for similar vertex pairs u and v, removal of (u, v) will incur a nonnegative cost. Likewise, for distant u and v pairs, addition of (u, v) will incur a nonnegative cost.

Consequently, the cost to transform the initial graph $G = (V, E)$ into a graph $G' = (V, E')$ is defined as follows:

$$cost(G \rightarrow G') = \sum_{(u,v) \in E \setminus E'} c^-(u, v) + \sum_{(u,v) \in E' \setminus E} c^+(u, v).$$

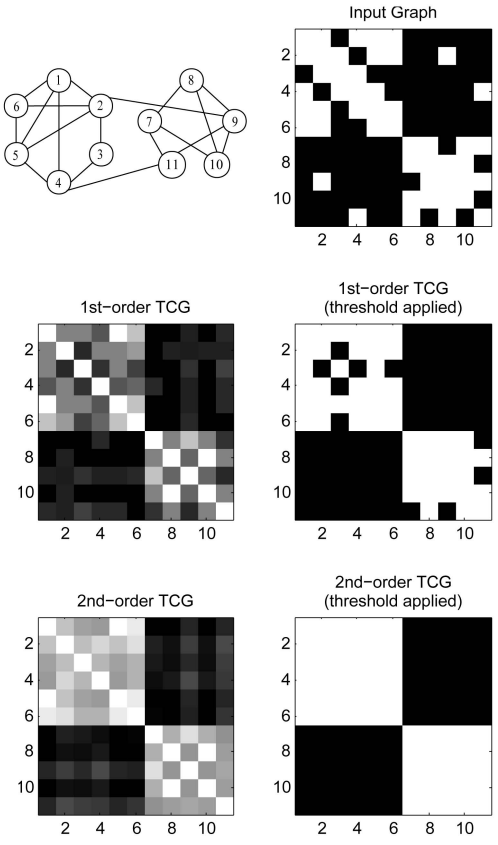

Fig. 1. Illustration of coconnectedness-based heuristic for an unweighted graph. Graph and its adjacency matrix as an image are shown in the top panel. Color scale is from 0 (black) to 1 (white). In the second and third panels, TCG^1 and TCG^2 are shown, respectively. The threshold-applied and binarized version of each is also shown to indicate that TCG gets closer to the underlying clique structure in each iteration.

2.2 Problem Statement

Given a similarity function $s : V^2 \rightarrow R^+$ and a weighted undirected graph $G = (V, E, s)$, find a union of cliques graph G^* such that $cost(G \rightarrow G^*) = \min\{cost(G \rightarrow G' | G' \text{ is a union of disjoint cliques})$.

In our method to tackle this problem, we do not explicitly use the cost function, but use it as an evaluation criterion for our simulation results. We show that the output of our algorithm minimizes the cost function when the error rate is low, and its corresponding cost is close to the optimal when the error rate is high. (See Section 5.3, Fig. 5.) As noted earlier, algorithms with explicit theoretical guarantee on performance are intractable for the large data sets. Further, assuming an underlying "corrupted-clique" model to explain real data, the method is guaranteed (Section 3) to improve in each iteration. This provides a theoretical foundation for its performance on real data.

3 Coconnectedness-Based Heuristic

In this section, we propose a heuristic-algorithm-based coconnectedness to tackle weighted cluster editing problem. Co-connectedness is described as the fraction of neighbors shared by the pair, and has been used in many different graph-theoretical problems [10], [11]. To illustrate using an example, consider Fig. 1. The underlying clique structure of

the input graph has two independent cliques formed by the vertices $1,2,3,4,5,6$ and $7,8,9,10,11$. The observed graph has some noise as some of the edges within the cliques are missing and there are some edges between the cliques.

For any vertex u, define the neighborhood vector $\vec{u}$ as the bit vector describing the set of neighbors. For example, $\vec{1} = [1,1,0,1,1,1,0,0,0,0,0]$. We use the Tanimoto coefficient (TC) to describe coconnectedness of two vertices u and v as follows:

$$TC(u,v) = \frac{\vec{u}\cdot\vec{v}}{\vec{u}\cdot\vec{u}+\vec{v}\cdot\vec{v}-\vec{u}\cdot\vec{v}}. \quad (1)$$

Note that the TC here is identical to the *Jaccard Coefficient* ($JC(u,v) = \frac{|N_u \cap N_v|}{|N_u \cup N_v|}$), but also works for weighted graphs (Section 4.2). It measures not only if two vertices are connected to the similar set of vertices but also if they are connected with similar edge weights.

In Fig. 1, for visualization purposes, we display the adjacency matrices of the graphs as images. Observe that for a spurious edge $(2,9)$, $\mathrm{TC}(2,9)$ is low, while for the missing edge $(2,4)$, $\mathrm{TC}(2,4)$ is high. Therefore, computing TC, and applying a threshold, we get a new sparse graph, TC graph, with fewer errors. Within two iterations, the graph reverts to a collection of two cliques. Note that TC of two vertices does not have direct interpretation of their original edge weight. It is possible for two vertices with high edge weight to have low TC and for two vertices with low edge weight to have high TC. We treat a high edge weight as FP if measured TC is low. Similarly, if the measured TC is high, we treat a low edge weight as FN.

We can generalize this idea and show calculations that help to show convergence. Following Ben-Dor et al. [6], define a (d,α)-*corrupted clique graph* as a collection of disjoint cliques, each of size d, in which intraclique edges are removed independently with probability α', and interclique edges are independently added with probability α' for an arbitrary $\alpha' \in [0,\alpha]$.

Let G be a (d,α)-corrupted-clique graph. If

$$\alpha < \min\left\{\frac{\ln d}{3d}, \frac{d}{n}\right\},$$

we can show that there exists a *tcg-threshold* such that the TC graph (TCG) is a (d,α')-corrupted-clique graph with $\alpha' < \alpha$. Note that if this is true, then, in each iteration, we will converge toward the underlying clique structure. In practice, we only need to do this for a few iterations and then output the connected components.

Consider two vertices u,v from the same clique. Then, assuming that edges in the clique are missing with probability α

$$E(\vec{u}\cdot\vec{v}) \geq (1-\alpha)^2 d.$$

The inequality is because some spurious edges may add to the dot product. Note that the dot product can be computed as the sum of d independent binary variables, each of which is 1 with probability $(1-\alpha)^2$. Consequently, we can use Chernoff's bound (e.g., [12]) to compute deviations from the mean as

$$Pr[\vec{u}\cdot\vec{v} < (1-\delta)(1-\alpha)^2 d] \leq e^{-\delta^2(1-\alpha)^2 d/2}, \quad (2)$$

for any $\delta > 0$. Correspondingly, for vertices u,v from different cliques

$$E(\vec{u}\cdot\vec{v}) = 2\alpha(1-\alpha)d + \alpha^2(n-2d) \leq 3\alpha d,$$

and, an equality similar to (2) holds

$$Pr[\vec{u}\cdot\vec{v} > (1+\delta)3\alpha d] \leq e^{-\delta^2 3\alpha d/4}. \quad (3)$$

Finally, we have

$$E(\vec{u}\cdot\vec{u}) = (1-\alpha)d + \alpha(n-d).$$

The condition $\alpha \leq d/n$ implies that

$$(1-\alpha)d \leq E(\vec{u}\cdot\vec{u}) \leq (2-\alpha)d,$$

and, with high probability

$$(1-\delta)(1-\alpha)d \leq \vec{u}\cdot\vec{u} \leq (1+\delta)(2-\alpha)d.$$

Finally, consider the expression $\vec{u}\cdot\vec{u}+\vec{v}\cdot\vec{v}-\vec{u}\cdot\vec{v}$. Using the Cauchy-Schwartz inequality, $\vec{u}\cdot\vec{v} \leq \max\{\vec{u}\cdot\vec{u}, \vec{v}\cdot\vec{v}\}$. Therefore, with high probability

$$(1-\delta)(1-\alpha)d \leq \vec{u}\cdot\vec{u}+\vec{v}\cdot\vec{v}-\vec{u}\cdot\vec{v} \leq 2(1+\delta)(2-\alpha)d.$$

Thus, if u,v are in the same clique, with high probability

$$\begin{aligned} TC(u,v) &\geq \frac{\vec{u}\cdot\vec{v}}{\vec{u}\cdot\vec{u}+\vec{v}\cdot\vec{v}} \\ &\geq \frac{(1-\delta)(1-\alpha)^2}{2(1+\delta)(2-\alpha)}. \end{aligned}$$

If u,v are in different cliques, then with high probability

$$TC(u,v) \leq \frac{(1+\delta)3\alpha}{(1-\delta)(1-\alpha)}.$$

If we can choose a threshold in between these two numbers, then, with high probability, we will get rid of the spurious edges, and add missing edges. This imposes the following condition on δ:

$$\frac{(1+\delta)3\alpha}{(1-\delta)(1-\alpha)} \leq \frac{(1-\delta)(1-\alpha)^2}{2(1+\delta)(2-\alpha)},$$

implying

$$\frac{(1+\delta)^2}{(1-\delta)^2} \leq \frac{(1-\alpha)^3}{6\alpha(2-\alpha)},$$

or

$$\delta \geq \frac{(1-\alpha)^{3/2} - (6\alpha(2-\alpha))^{1/2}}{(1-\alpha)^{3/2}(6\alpha(2-\alpha))^{1/2}}. \quad (4)$$

Finally, we require that the probability that a spurious edge remains in the TCG, or a true edge is missing is lower than α. This is bounded using (2) and (3) to be

$$4e^{-\delta^2 3\alpha d/4} < \alpha,$$

implying

$$\delta^2 < \frac{4\ln(4/\alpha)}{3d\alpha}. \quad (5)$$

Equations (4) and (5) are satisfied when $\alpha < \frac{\ln d}{3d}$, implying that error is reduced in a single iteration of TCG generation. In practice, our approach works well for much higher error rates ($\alpha < 0.5$), with results comparable to CAST, but extending to larger data sets.

3.1 Running Time Analysis

If the average degree of vertex is d, computation of TC for a pair of vertices costs $O(d)$. In each iteration of TCG generation, we compute TC for only pairs with distance at most 2, since otherwise TC will be zero. Thus, the running time complexity of the algorithm is $O(|V|d^3)$ per iteration and scales linearly with the number of iterations.

The number of iterations, K, depends on the size of the underlying cliques and the error rate. At a fixed error rate the neighborhood similarity for a pair of vertices in a small clique is more influenced than a pair of vertices in a large clique. Thus, it takes more iterations for smaller cliques to edit all FP and FN edges. In our experiments, at error rate $\epsilon = 0.2$, in a corrupted graph of 640 vertices, we recover the cliques of size 128 in TCG^1, while we recover the cliques of size 16 in TCG^3. (See Fig. 4.)

4 TCLUST

Clustering of large gene expression data sets is challenging. Our clustering method TCLUST is based on two ideas. The first is an assumption that the expression clusters resemble corrupted cliques, and coconnectedness can be used for clustering. This, of course, cannot be proved. However, we evaluate our clustering results using independent biologically meaningful metrics, and show that whether the underlying model is true or not, we can use coconnectedness.

The second idea, motivated in part by Gibson et al. [13], is based on fingerprinting which suggests that coconnectedness can be exploited looking only at a subset of genes to which a particular gene is highly correlated. As this subset is small, our algorithm can comfortably handle large input sizes. Any loss of performance is handled by repeated iterations of the coconnectedness heuristic from Section 3.

Fig. 2 provides an overview of TCLUST. A fingerprinting strategy is used to generate a correlation coefficient graph (CCG) from gene expression data. This is followed by iterative computation of TCG until connected components of the TCG are clique-like. Note that, once a connected component is dense enough, it will only get denser in subsequent iterations and no two disjoint connected components will be merged. Thus, it is possible to output a connected component as a cluster at any iteration as its edge density, i.e., number of edges/number of pairs exceeds a threshold.

While CAST and TCLUST both have similar goal of identifying underlying cliques, TCLUST should converge faster as vertex pairs with strong (respectively, weak) coconnectedness are quickly identified as such, and remain as edges (respectively, nonedges) for the remainder of the iterations. Specifically, if the two vertices end up in different connected components, their TC will always be 0, and they will never converge into one component. This implies that we only need to compute TC for vertex pairs in the same connected component, and a few iterations should suffice to establish the clusters.

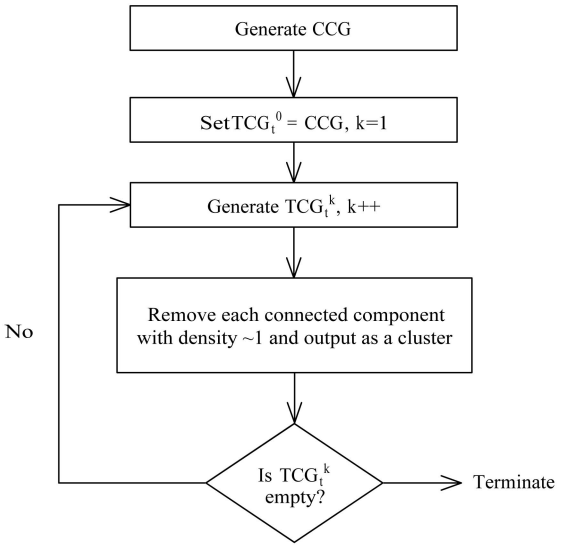

Fig. 2. Flow chart for the TCLUST algorithm.

We apply our method on a mouse gene expression data set with 45,101 probe sets and 295 samples. (See the Appendix for details.) In our experiments, we show that TCLUST is very efficient while it is still able to detect biologically meaningful gene groups.

In the following sections, we discuss the different steps spelled out in the flowchart in detail. In Section 5, we demonstrate the performance of our algorithm on simulated data. In Section 6, we compare the performance of TCLUST, CAST, and K-means on filtered expression data. Finally, in Section 6.3, we apply TCLUST to our large gene expression data set for gene set enrichment analysis.

4.1 Generating Correlation Coefficient Graph

We use the Pearson's correlation coefficient as a measure of coexpression of genes/probe sets. Let x_{ik} denote the expression level of p_i in the Kth sample. For a pair of probes p_i, p_j

$$s(p_i, p_j) = \frac{\sum_{k=1}^{n}(x_{ik} - \bar{x}_i)(x_{jk} - \bar{x}_j)}{\sqrt{\sum_{k=1}^{n}(x_{ik} - \bar{x}_i)^2 \sum_{k=1}^{n}(x_{jk} - \bar{x}_j)^2}}.$$

Clearly $s(p_i, p_j) = s(p_j, p_i)$. Define CCG as a complete undirected, weighted graph (V, E, s), with s defining edge weights. We use a threshold on the weights, only including edges that exceed the threshold. Other results have suggested that rank-based correlations are more robust to noise. However, in our own experiments, the data set was of sufficiently high quality that we continued with Pearson's correlation, which was required by our collaborators on the project.

It is a common strategy to reduce the complexity of CCG by discarding the edges with edge weights smaller than a chosen threshold. In our experiments, we observe that correlation coefficients of a probe set with the rest of the probe sets follow a unimodal distribution with a small tail. (See Fig. 3 for the distribution of four randomly chosen probe sets.) However, the shape of the distribution differs from probe set to probe set. This makes it difficult to set a fixed threshold that would not be biased for any probe set.

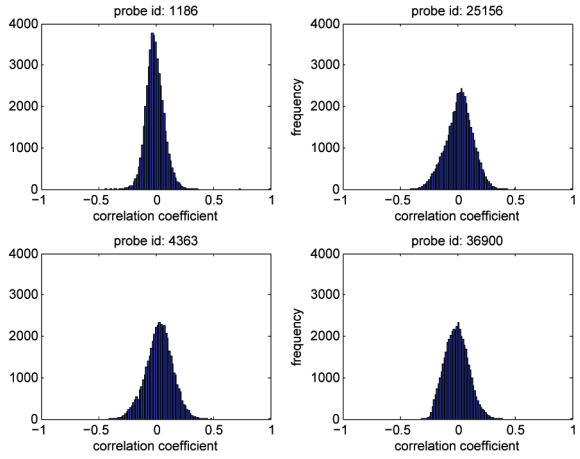

Fig. 3. Distribution of correlation coefficients with the rest of $45K$ probe sets for four randomly chosen probe sets.

We use a novel *per-vertex-threshold* scheme to determine edges. First, define the *top-neighbors* of vertex p_i, $N(p_i)$ as follows: let μ_i and σ_i be the mean and standard deviation of the correlation coefficients between vertex p_i and all other vertices in V, respectively. Denote $p_j \in N_\theta(p_i)$ if and only if

$$\mathrm{s}(p_i, p_j) \geq \mu_i + \theta\sigma_i.$$

Define $\mathrm{CCG}_\theta = (V, E_\theta, s)$, where V is the set of probe sets, and

$$E_\theta = \{(p_i, p_j) | p_i \in N_\theta(p_j), p_j \in N_\theta(p_i)\}.$$

Per-vertex thresholding can be thought as a fingerprinting scheme where each vertex/probe set is fingerprinted with the set of its top neighbors. It keeps the graph sparse, and also controls for certain vertices (genes) having higher overall expression values compared to other genes. We use CCG_3 (determined empirically) as the input to TCLUST. However, note that no assumption is made w.r.t the distribution of scores of correlated genes.

We reiterate that the sparsification by edge and node thresholding is provided as an option which allows us to handle large graphs. It is neither required nor assumed that the input is sparse. Indeed, while evaluating cluster quality, we compare against the original (complete) graph. It is possible that the sparsification could reduce the quality of our clusters. However, the coconnectedness property is mostly preserved by sparsification, and our results show that there is no loss of quality by introducing this.

4.2 Generating Tanimoto Coefficient Graph (TCG)

The TC is a measure of similarity between two real-valued vectors, defined by (1). For a weighted graph, $G = (V, E, w)$, denote

$$\vec{W}_i = [w(p_i, p_j) : p_j \in V]$$

as the vector of edge weights incident on vertex p_i. We define the TC between a pair of vertices (p_i, p_j) as follows:

$$TC(p_i, p_j) = TC(\vec{W}_i, \vec{W}_j).$$

This reweighing of edges allows us to define a family of *Tanimoto Coefficient Graphs* TCG_t^k for arbitrary $k \in Z^+$, and a real-valued tcg-threshold t. Specifically, $\mathrm{TCG}_t^0 = (V, E, w)$ and $\vec{W}_i^0 = \vec{W}_i$.

For $k > 0$, $\mathrm{TCG}_t^k = (V, E^k, w^k)$, where

$$E^k = \left\{(p_i, p_j) | \mathrm{TC}(\vec{W}_i^{k-1}, \vec{W}_j^{k-1}) \geq t, \forall p_i, p_j \in V\right\},$$

$$w^k(p_i, p_j) = \begin{cases} \mathrm{TC}(\vec{W}_i^{k-1}, \vec{W}_j^{k-1}), & (p_i, p_j) \in E^k, \\ 0, & \text{otherwise.} \end{cases}$$

As we iterate over TCG^K for $K = 0, 1, \ldots,$ the connected components should resemble the underlying clique structure. Therefore, we output the connected components as the final clusters.

4.3 Implementation of TCLUST

We implemented the core algorithm in C++. The input to the program is the CC graph, a tcg-threshold $t \in R$, and $k \in Z^+$, the number of iterations. (See Fig. 2.)

We use R [14] statistics package to generate the CCG for input to TCLUST. We compute the correlation matrix and filter the matrix using per-vertex thresholding technique.

5 Results on Simulated Data

5.1 Corrupted Random-Weighted Graph Model

Ben-Dor et al. [6] proposed a natural corrupted random graph model to test the performance of CAST in retrieving the underlying clique structure from "corrupted" graphs.

We extend their model to include weighted graphs. The Corrupted Random-Weighted Graph (CRWG) model has three input parameters (S, T, ε) defined as follows: S is the underlying clique structure denoted by the size of the underlying cliques. As an example, $S = [2 \times 8, 2 \times 16, 1 \times 32]$ denotes a structure consisting of two cliques of size 8, two cliques of size of 16, and one clique of size 32. T is the intraclique threshold, i.e., any edge weight within a clique is larger than T and any edge weight between cliques is less than or equal to T. Finally, ε is the error range of the observed edge weights.

In the ideal case, we assume a weighted complete graph with an underlying clique structure where $\varepsilon = 0$. Therefore, when a threshold T is applied, we can retrieve the cliques. In CRWG model, this ideal case is corrupted. Each edge weight w is replaced by a random weight uniformly distributed in the range $[w - \varepsilon, w + \varepsilon]$. Thus, when the threshold T is applied on the observed graph, an interclique edge may appear since $(w + \varepsilon)$ may exceed t, and an intraclique edge may not appear since $(w - \varepsilon)$ may be smaller than T. The effect of changing weight is the weighted analog of adding spurious edges, or deleting true edges in the corrupted-clique model. An edge with actual weight $T < w \leq T + \epsilon$ is not in the thresholded observed graph G with probability $\frac{T-w+\epsilon}{2\epsilon} \in [0, 0.5)$. Similarly, an edge with actual weight $T - \epsilon < w < T$ is in G with probability $\frac{w+\epsilon-T}{2\epsilon} \in (0, 0.5)$.

The output of TCLUST is a set of clusters. In a corrupted graph model, it is reasonable to think of each cluster as a complete subgraph. However, data sets may only have underlying dense subgraphs which are not complete. Therefore, we can use either the TCG^k itself as the output or we can use the collection of cliques induced by the

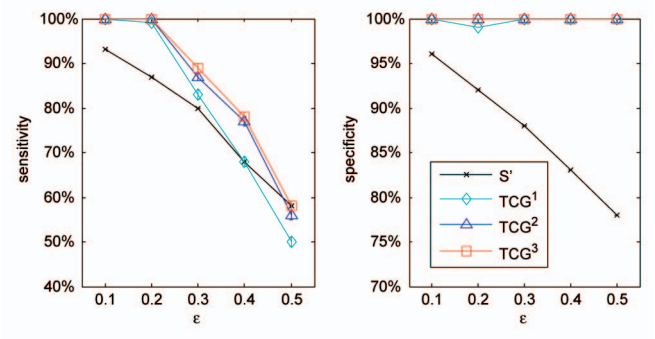

Fig. 4. Comparison of underlying clique structure and filtered TCG^1, TCG^2, and TCG^3. Average cost, sensitivity, and specificity are plotted with error bars. Random graphs are generated by the CRWG model with parameters $\mathrm{S} = [16 \times 8, 8 \times 16, 4 \times 32, 2 \times 64, 1 \times 128]$, $T = 0.6$, and ε ranging from 0.1 to 0.5. TCG^k is the kth-order TCG computed from S' iteratively. Sensitivity $= TP/(FN + TP)$ and specificity $= TN/(FP + TN)$ are computed according to S, where TP, FP, TN, and FN are defined based on the existence of edges after applying the tcg-threshold on the generated TCG. Tcg-threshold is set to its optimal values among $[0.1, 0.2, \ldots, 0.9]$.

vertices of each connected component of TCG^k as the output. We test both of these outputs in the following sections. In the second case, we also compare against a version of CAST, reimplemented for the weighted case.

5.2 Recovering Underlying Structure by TCG

To test the performance of iterative computation of TCG in recovering the underlying cliques, we generate 20 random graphs using the CWRG model with $S = [16 \times 8, 8 \times 16, 4 \times 32, 2 \times 64, 1 \times 128]$, $T = 0.6$, and $0.1 \leq \varepsilon \leq 0.5$. Note that each graph has 31 underlying cliques of sizes $8, 16, 32, 64, 128$ and the number of vertices participating in different size cliques is the same. The input graph G to our clustering algorithm is the threshold graph that is obtained by filtering the edges with weights smaller than $T = 0.6$. Thus, G has a number of extra and missing edges depending on the error rate ϵ.

If an edge in TCG is indeed between two clique members, it is considered as a TP; otherwise, an FP. TN and FN are defined accordingly. We use two similarity measures: *sensitivity* and *specificity*. We define sensitivity as $TP/(FN + TP)$ and specificity as $TN/(FP + TN)$.

Fig. 4 shows our simulation results for the comparison of underlying clique structure S with filtered TCG^1, TCG^2, and TCG^3 by applying tcg-threshold. The tcg-threshold is varied between 0.1 and 0.9 to select the setting with better sensitivity and specificity results. In Fig. 4, the results with only the best settings are shown.

The results show that iteratively generating TCG allows us to get closer to the underlying clique structure S. For example, in the case where $\varepsilon = 0.2$, sensitivity and specificity values for G are improved from (87 percent, 92 percent) to (99 percent, 99 percent) by TCG^1 and to (100 percent, 100 percent) by TCG^2 revealing S exactly. In each case, we improve significantly upon the corrupted graph.

5.3 CAST versus TCLUST

In this section, we compare the performance of TCLUST with CAST [6]. As the code of CAST was not available, and we need a version that works for weighted graphs, we implemented TCLUST and CAST algorithms in C++ using the same framework. While there is the real caveat of comparing our own implementation of CAST, we took care to maintain the fidelity of the approach. Indeed, the project started out by attempting to use CAST to cluster the data sets. However, in the interest of fairness, the results below are best interpreted by considering the CAST running times generically. Both return a collection of cliques. We evaluate the quality of the output clique graphs by comparing their associated cost from the input graph with the cost of obtaining the underlying clique structure S.

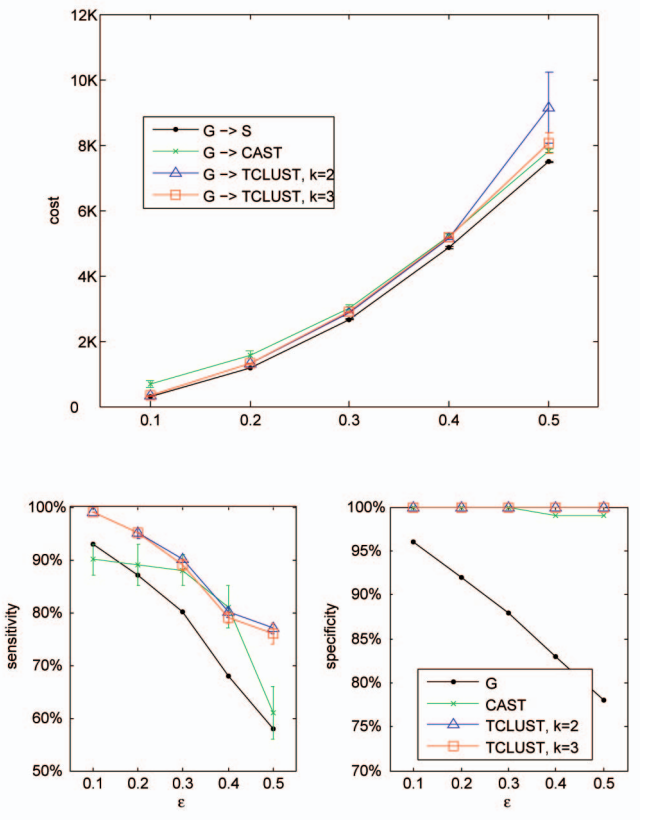

Fig. 5. Comparison of CAST and TCLUST with $K = 2, 3$. Random graphs are generated by the CRWG model with parameters $\mathrm{S} = [16 \times 8, 8 \times 16, 4 \times 32, 2 \times 64, 1 \times 128]$, $T = 0.6$, and ε ranging from 0.1 to 0.5. Sensitivity $= TN/(FP + TN)$ and specificity $= TP/(FN + TP)$ are computed according to S. While computing sensitivity and specificity, clusters obtained by the method are treated as a clique. Affinity-threshold and tcg-threshold are set to 0.5 and 0.2, respectively, which are their best settings among the values $[0.1, 0.2, \ldots, 0.9]$.

We compare the results for CAST and TCLUST of degree $K = 2, 3$. Both CAST and TCLUST take a parameter as input: affinity-threshold and tcg-threshold, respectively. We vary both parameters from 0.1 to 0.9. In Fig. 5, we show the simulation results of CAST and TCLUST with only the setting which is optimal in terms of cost.

This simulation shows that both CAST and TCLUST closely approximate the cost of the underlying clique structure even with high error rate. However, TCLUST has better sensitivity and specificity results with a much lower standard deviation at each error rate. Note that each connected component is treated as a complete graph in this test. This changes the FP, and FN rate, and so the results in Figs. 4 and 5 are not directly comparable.

6 Comparison of Different Clustering Methods on Microarray Expression Data

We compare TCLUST with two existing methods: K-means and CAST on microarray data. We explore the performance of these methods over a wide range of their parameters (K, affinity-threshold, tcg-threshold). We limit the data set to 5,000 probe sets with the highest variation across samples, as K-means and CAST did not converge for larger sizes. In the next section, we will present results of TCLUST on the complete data set ($45K$ probes).

We use a weighted version of CAST for comparison [6]. We applied a prefilter to the pairwise similarity matrix, discarding all edges less than 0.3. In the absence of a prefilter, CAST does not converge and outputs only five clusters of almost equal sizes, none of which were functionally enriched.

6.1 Functional Enrichment

In clustering genes according to the expression data, a common goal is to cluster functionally related genes, and we use this as a test for the three methods. We extracted all *functional gene sets* (FGS) of size ≤ 500 from six different databases: KEGG pathways database (KEGG) [15], ingenuity pathways database (ING) [16], gene ontology (GO) database [17] in categories cellular component (CC), molecular function (MF), and biological process (BP), and mouse phenome database (MPD) [18]. The chosen 5,000 probe sets map to 3,776 genes, of which 23.36 percent are annotated in KEGG, 21.21 percent in ING, 31.89 percent in CC, 64.27 percent in MF, 58.16 percent in BP, and 30.27 percent in MPD.

The first step is to decide if a predicted cluster C is functionally enriched in an FGS F. We compute a p-value for enrichment of F in C, using a hypergeometric test [19]. For details, see Appendix A3. We set the significance threshold to $p \leq 0.001$. Also, we say that a cluster is functionally enriched if it is significant for at least one FGS.

As the parameters for clustering are changed, we get differing number/sizes of clusters. Larger cluster sizes include more genes, but are less likely to be enriched in a single FGS. Therefore, we use *gene coverage*—defined as the fraction of genes in functionally enriched clusters—to compare the different methods. However, some of the functional databases may not include all genes. Therefore, a related measure is *db-coverage*, defined as the fraction of genes annotated in the specific database that end up in a functionally enriched cluster.

In Fig. 6, we compare TCLUST, CAST, and K-means in terms of gene coverage and db coverage. For each clustering method, we show seven different parameter settings exploring a wide range. The *tcg*-threshold for TCLUST varies from 0.2 to 0.8, the affinity threshold for CAST varies from 0.3 to 0.9, and the number of clusters for K-means varies from 200 to 3,000 so that all three methods give similar ranges for the number of clusters. Each choice of a method, and a parameter setting is plotted according to its gene, and db-coverage. The best parameter settings for all three methods are highlighted by larger data points and labels in the plots.

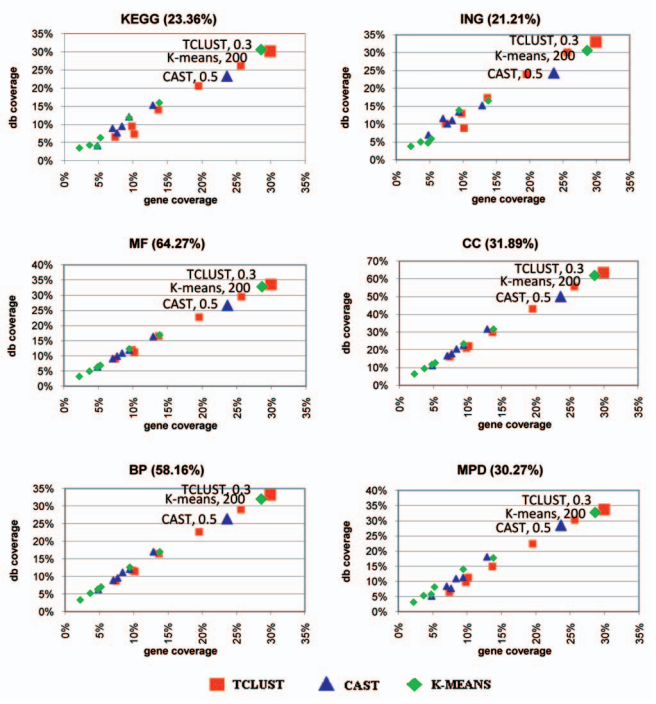

Fig. 6. Comparison of TCLUST, CAST, and K-means in terms of gene and database coverage in the functionally enriched clusters. For each clustering method, seven different parameter settings are shown. The number of clusters for K-means varies from 200 to 3,000, affinity-threshold for CAST varies from 0.3 to 0.9, and tcg-threshold for TCLUST varies from 0.2 to 0.8. The best settings for TCLUST, CAST, and K-means (tcg-threshold$= 0.3$, *aff-threshold*$= 0.5$, and $K = 200$) are highlighted by larger data points in the plots.

The results show that TCLUST does generally better than CAST and K-means. The best setting for TCLUST has better gene, and db-coverage compared to best setting of CAST and K-means, and also many other settings show high coverage. Note that the gene coverage in the functionally enriched clusters is limited to 30 percent for all methods because of incomplete gene annotation.

We use a false discovery rate (FDR) approach to evaluate the significance of the number of functionally enriched clusters obtained by the clustering methods. We picked the best parameter setting that achieves the highest database coverage for each method. We computed many random permutations of gene annotation labels. Each method/parameter setting pair is applied to the randomized data in which the number and the size distribution of the clusters remain the same. Any enriched cluster discovered in a randomized data is a false positive. The ratio of the average number of false positives to the actual number of functionally enriched clusters describes the FDR.

In Table 1, we give the db-coverage, the number of functionally enriched clusters, and the associated FDR results for the clusterings obtained by TCLUST, CAST, and K-means with their best parameter settings. As we see, FDR for TCLUST is substantially lower than the FDR for CAST and K-means while it achieves better db-coverage and higher number of functionally enriched clusters. This observation suggests that larger database coverage of TCLUST is not due to the size distribution or number of the clusters.

TABLE 1
False Discovery Rates Associated with the Number of Functionally Enriched Clusters (#fec) for TCLUST, CAST, and K-Means

	K-means			CAST		
	db-coverage	#fec	FDR	db-coverage	#fec	FDR
KEGG	30.61%	2	78.6%	23.36%	13	6.2%
ING	30.59%	2	59.4%	24.34%	8	8.9%
CC	61.96%	8	36.6%	50.08%	9	15.5%
MF	32.80%	11	51.5%	26.70%	17	14.4%
BP	31.97%	12	40.0%	26.41%	23	12.6%
MPD	32.72%	10	40.6%	28.52%	11	25.9%

	TCLUST		
	db-coverage	#fec	FDR
KEGG	30.16%	13	5.9%
ING	32.96%	8	8.8%
CC	63.46%	13	8.3%
MF	33.54%	24	7.6%
BP	33.15%	26	8.6%
MPD	33.68%	14	14.9%

The parameter setting which is best in terms of gene and database coverage is used for each of the clustering method. At high gene and database coverage, TCLUST still has lower FDR.

TABLE 2
Timing Results in Seconds as a Function of N, the Number of Input Probe Sets to Cluster for TCLUST, CAST, and K-Means

N	K-means	CAST		
		w/o cutoff	w/ cutoff	per-vertex threshold
100	0.02	0.32	0.30	0.04
500	1.08	138	7.06	0.11
$1K$	9.31	2144	35.18	0.69
$2K$	71.91	7653	113.36	3.41
$5K$	1392	444405*	3442	91.64
$10K$	19693	-	8240	526
$20K$	-	-	-	5220
$45K$	-	-	-	109847

N	TCLUST		
	k=0	k=1	k=2
100	0.01	0.04	0.06
500	0.02	0.12	0.17
$1K$	0.04	0.45	0.65
$2K$	0.10	1.91	2.59
$5K$	0.32	18.43	24.30
$10K$	1.44	159	203
$20K$	5.93	1125	1363
$45K$	49.08	13223	15676

The parameters for the methods are chosen as the setting that performed best on the 5,000 probe sets in terms of gene and database coverage (aff. threshold=0.5, tcg-threshold=0.3, k = $N/25$). We give two timing results for CAST with and without a cut-off. Originally, CAST takes the whole similarity matrix as input. We also assess the time by applying a cut-off=0.3 on the similarity matrix, since as the graph gets larger CAST becomes intractable. For TCLUST, we give three timing results varying the degree K of the algorithm from 0 to 2. (CAST does not converge and reaches the upper limit for the number of iterations.)*

6.2 Timing

In this section, we compare K-means, CAST, and TCLUST with $K = 0, 1, 2$ in terms of timing for different numbers of probe sets to cluster. (See Table 2.) K-means takes the whole similarity matrix as input while TCLUST and CAST take an input graph using similarities. The timing for generation of the similarity matrix and the input graphs is excluded and only the time required for the algorithm itself is given. For TCLUST, we give three timing results varying the degree K of the algorithm from 0 to 2. We give three timing results also for CAST, running on the complete, filtered at a cut-off, and per-node thresholded similarity graphs. Originally, CAST takes the whole similarity matrix as input. However, as the graph gets larger, CAST becomes intractable. Thus, we assess also the timing for CAST after applying a cut-off=0.3 on the similarity matrix. In order to clarify gain in running time by per-node thresholding, we also report running time of CAST on the per-node thresholded similarity graph.

The number of probe sets is varied from 100 to $10K$ by selecting the probe sets with the highest variation across the samples. The timing table was extended until $N = 45K$ for TCLUST only as it is not feasible to run K-means and CAST on large data sets. The parameters for the methods are chosen as the setting that performed best on the 5,000 probe sets in terms of gene and database coverage in Section 6.1 (aff. $\mathrm{threshold} = 0.5, \mathrm{tcg\text{-}threshold} = 0.3, \mathrm{k} = \mathrm{N}/25$). The programs were run on machine with 32 GB memory and a single 2.2 GHz AMD Opteron Processor.

Results clearly indicate the speed advantage of TCLUST.

6.3 Results on Complete Data Set

We ran TCLUST on the entire corpus of expression data with the goal of identifying functionally related gene sets using tcg-threshold=0.3. This threshold was optimized (as described earlier) by using a reduced data set of randomly chosen 5,000 probes (out of the 45,000 probes). As different data sets vary in quality, such an optimization is desirable. It is also worth noting that 1) other tools, like CAST, do not provide any guidance on how to set parameters, and 2) our tool, when run with default parameters, does not do much worse on the data sets that we have tried (data not shown).

All 45,101 probe sets mapping onto 21,452 genes were clustered. Of the 25,172 resulting clusters, majority were singletons, and 550 had size in range $[3, 500]$ covering 5,994 probe sets. The clusters in size $[3, 500]$ were mapped to Entrez gene IDs and then analyzed in terms of functional enrichment in each of the gene annotation databases. There were 32 out of the 550 clusters found to be functionally enriched at $p \leq 0.001$.

A cluster can be seen as a collection of two types of members: annotated and unannotated. The annotated genes are used to detect the FGSs in which the cluster is enriched in. This information can be used as a quality measure for the cluster and also as a basis for function prediction of the unannotated members.

Among the clusters with the most significant functional enrichments was one set of 231 genes dominated with genes of muscle-related functions. It was functionally enriched in at least one FGS in each database: muscle contraction (BP, $p < 10^{-37}$), contractile fiber (CC, $p < 10^{-39}$), structural constituent of cytoskeleton (MF, $p < 10^{-14}$), and calcium signaling pathway (KEGG, $p < 10^{-25}$). There were 181 genes in this cluster which were annotated in at least one of the enriched categories. The enrichment in muscle contraction is defined by 28 genes which are primarily in the actin, actinin, myosin, tropomyosin, and troponin protein families, genes which are well known to play important roles in muscle tissue [20]. This enrichment is also supported by 32 members of the actin, actinin, myosin, and troponin protein families which had no previously annotated role in muscle function. In addition, there are also some members of this cluster which also had no

annotated role in muscle function but have been previously discussed in literature. For example, the expression of leiomodin 3 (Lmod3) shows high correlation with the other members of the muscle contraction cluster, suggesting that it might have a related function. This hypothesis is further supported by known roles for the other leiomodin family members, Lmod1 and Lmod2, in smooth muscle and cardiac muscle function, respectively [21].

Another cluster we detected has 36 genes and it is enriched in genes involved in hormone activity (MF, $p < 10^{-13}$), and cysteine-type endopeptidase activity (MF, $p < 10^{-3}$). Nine of the genes are annotated in hormone activity, and three genes are annotated in cysteine-type endopeptidase activity. Among the other unannotated genes in this cluster are cathepsin 3 (Cts3). A putative role for Cts3 in hormone activity and peptidase function is supported by evidence for extracellularly acting cathepsins mediating thyroid hormone liberation in thyroid epithelial cells [22].

Although the examples highlighted here represent only two of the many functionally enriched clusters discovered, they illustrate the tremendous potential of functional inference based on genomewide clustering.

7 Conclusion

We introduce here a new method, TCLUST, for clustering large, genome-scale data sets. The algorithm is based on measures of coconnectedness to identify dense subgraphs present in the data. We have applied this method to a large reference gene expression data set, and showed that the resulting clusters show strong enrichment in known biological pathways.

Although TCLUST has been shown to perform as good as or better than existing methodologies, as with any methodology, certain caveats must be noted. A possible shortcoming might be that once two vertices end up in different clusters, they are never reconnected. On the one hand, this makes the algorithm converge faster; on the other hand, it might lead to some loss of sensitivity for higher error rates. In principle, this could be adjusted, by applying the tcg-thresholds more judiciously, gaining some FN edges at the cost of some FP edges, and increasing the number of iterations. We will explore this, and similar directions in future research. Also, the theoretical justification will be clarified and extended to more general settings. Specifically, while we present sufficient conditions on error rate α, we do not report any necessary conditions. These will be the subject of future investigation.

While the development of TCLUST was motivated by the lack of a suitable clustering tool for large gene expression data sets, its performance on smaller data sets is superior to or at least, competitive with established methods. Moreover, the method is based on the relatively broad assumptions that the clusters behave like dense subgraphs of an appropriate sparse subgraph. Therefore, TCLUST should be applicable in a variety of biological settings, and offers a new approach, complementing existing methods. As biological data sets continue to grow in scale, the importance of efficient algorithms for clustering genome scale will become paramount, requiring continued development of efficient algorithms.

TABLE 3
Eleven Tissues Dissected from 29 Strains: Adipose (AD), Amygdala (AM), Dorsal Root Ganglia (DR), Frontal Cortex (FC), Hippocampus (HC), Hypothalamus (HT), Liver (LV), Nucleus (NC), Pituitary (PT), Skeletal Muscle (SM), Spleen (SP)

strains	tissues										
	AD	AM	DR	FC	HC	HT	LV	NC	PT	SM	SP
129S1/SvImJ		*	*	*	*	*	*	*	*	*	*
A/J	*	*	*	*	*	*	*	*	*	*	*
AKR/J	*	*	*	*	*	*	*	*	*	*	*
BALB/cByJ	*	*	*	*	*	*	*	*	*	*	*
BTBR T+ tf/J	*	*	*	*	*	*	*	*	*	*	*
BUB/BnJ	*	*	*	*	*	*	*	*	*	*	*
C3H/HeJ	*	*	*	*	*	*	*	*	*	*	*
C57BL/6J	*	*	*	*	*	*	*	*	*	*	*
C57BR/cdJ	*	*	*	*		*	*	*	*	*	*
C58/J	*	*	*	*	*	*	*	*	*	*	*
CBA/J	*	*	*	*	*	*	*	*	*	*	
CE/J		*		*	*	*	*	*	*	*	*
DBA/2J	*	*	*	*	*	*	*	*	*	*	*
FVB/NJ	*	*	*	*	*	*	*	*	*	*	*
I/LnJ	*	*		*	*	*	*	*	*	*	*
KK/HlJ	*	*	*	*	*	*	*	*	*	*	*
MA/MyJ	*	*		*	*	*	*	*	*	*	*
MRL/MpJ	*			*	*		*		*	*	
NOD/LtJ	*	*	*	*	*	*	*	*	*	*	*
NON/LtJ	*	*	*	*	*	*	*	*	*	*	*
NZO/HlLtJ	*			*	*		*			*	*
NZW/LacJ	*	*	*	*	*	*	*	*	*	*	*
P/J	*		*	*	*		*	*		*	*
PL/J	*	*	*	*	*	*	*	*	*	*	*
RIIIS/J	*	*	*	*	*	*	*	*	*	*	*
SJL/J	*	*	*	*		*	*	*	*		*
SM/J		*	*	*	*	*	*	*	*	*	*
SWR/J	*	*	*	*	*	*	*	*	*	*	*
WSB/EiJ		*		*	*	*	*	*	*	*	*
#strains	25	26	23	29	28	26	29	27	26	28	28

The samples for which gene expression data are available are indicated by "."*

Appendix

A.1 Expression Profiling and Preprocessing

Eleven tissues (adipose, amygdala, dorsal root ganglia, frontal cortex, hippocampus, hypothalamus, liver, nucleus accumbens, pituitary, skeletal muscle, and spleen) were dissected from a panel of 29 diverse inbred strains. Gene expression analysis was performed using Affymetrix MOE430v2 GeneChips. After samples which did not pass quality control were removed, data for 295 samples remained. (See Table 3.) Each microarray measured expression for 45,101 probe sets targeting 21,452 unique mouse genes. Each expression measurement was summarized by gcRMA (bioconductor package; [23], [24]) from the quantile-normalized probe intensities of a probe set.

We treat each set of the same tissue samples from up to 29 diverse inbred mouse strains as a separate tissue-specific data set. We preprocess each of these 11 data sets separately by centering the log-transformed intensity values at zero, to highlight the variation in expression across strains and emphasize genetic background. We then merge the data sets so that we have a single data set with 45,101 probe sets and 295 samples.

A.2 Biological Knowledge Represented in Gene Sets

The gene ontology (GO) database [17] was downloaded from http://www.geneontology.org/ontology/gene_ontology.obo. The snapshot of Apr. 03, 2006, was used in this analysis,

which contains 21,316 GO terms in three categories for biological process (BP), molecular function (MF), and cellular component (CC). Three unknown categories, "GO:0000004," "GO:0005554," and "GO:0008372", were removed for the analysis. The mapping from Entrez Gene IDs to GO terms was obtained from NCBI's gene2go table (3 Apr. 2006, snapshot from ftp://ftp.ncbi.nih.gov/gene/DATA/gene2-go.gz). In addition, we utilized two databases of manually annotated metabolic and signaling pathways. The KEGG pathway database [15] was downloaded from ftp://ftp.genome.jp/pub/kegg/pathways/mmu/. The snapshot of 26 Apr. 2006, was used, which contains 174 pathways for mouse. Ingenuity pathways database (ING) [16] was obtained from Ingenuity Systems, which contains 137 pathways for mouse. Finally, mouse phenome database (MPD) [18], which is a repository of phenotypic and genotypic data on diverse inbred strains was downloaded on 17 May 2007, from http://phenome.jax.org/phenome. All flat-file formatted databases were parsed by individual python scripts for use in the functional analysis.

A.3 Functional Analysis of Clusters

For each cluster and functional gene set (FGS) pair, the enrichment p-value p is calculated as follows:

$$p = \frac{\binom{N_F}{k}\binom{N-N_F}{n-k}}{\binom{N}{n}},$$

where N_F is defined as the number of genes assigned to the FGS, n is the number of genes in the cluster, K is the number of genes in the cluster that are annotated in the FGS, and N is the total number of genes [19].

A.4 Randomization Procedure for FDR Calculation

We compute the FDR associated with the number of functionally enriched clusters obtained by a clustering as follows: We generate 100 random clusterings with the same number of clusters and cluster size. This is achieved by simply permuting the probe set or gene labels. For each random clustering, the number of functionally enriched clusters was recorded so that we obtain a null distribution for the number of functionally enriched clusters. We compute the FDR for the actual number of functionally enriched clusters by

$$FDR(n) = \frac{n_R}{n},$$

where n is the actual number of functionally enriched clusters, and n_R is the mean number of functionally enriched clusters in the null distribution.

ACKNOWLEDGMENTS

This work was supported in part by a research gift, US National Science Foundation (NSF) IIS-0810905, from Glaxo SmithKline (Vineet Bafna, Banu Dost), and by an internship at The Genomics Institute of the Novartis Research Foundation (Banu Dost).

REFERENCES

[1] R. Shamir, R. Sharan, and D. Tsur, "Cluster Graph Modification Problems," *Discrete Applied Math.*, vol. 144, nos. 1/2, pp. 173-182, http://dx.doi.org/10.1016/j.dam.2004.01.007, 2004.

[2] S. Delvaux and L. Horsten, "On Best Transitive Approximations to Simple Graphs," *Acta Informatica*, vol. 40, no. 9, pp. 637-655, 2004.

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.

[4] J. Quackenbush, "Computational Analysis of Microarray Data," *Nature Rev. Genetics*, vol. 2, no. 6, pp. 418-427, http://dx.doi.org/10.1038/35076576, June 2001.

[5] M. Gerstein and R. Jansen, "The Current Excitement in Bioinformatics—Analysis of Whole Genome Expression Data: How Does It Related to Protein Structure and Function," citeseer.ist.psu.edu/gerstein00current.html, 2000.

[6] A. Ben-Dor, R. Shamir, and Z. Yakhini, "Clustering Gene Expression Patterns," *J. Computational Biology*, vol. 6, nos. 3/4, pp. 281-297, citeseer.ist.psu.edu/ben-dor99clustering.html, 1999.

[7] F.D. Gibbons and F.P. Roth, "Judging the Quality of Gene Expression-Based Clustering Methods Using Gene Annotation," *Genome Research*, vol. 12, no. 10, pp. 1574-1581, http://www.hubmed.org/display.cgi?uids=12368250, Oct. 2002.

[8] S. Rahmann, T. Wittkop, J. Baumbach, M. Martin, A. Truss, and S. Böcker, "Exact and Heuristic Algorithms for Weighted Cluster Editing," *Proc. Computational Systems Bioinformatics Conf*, vol. 6, pp. 391-401, http://view.ncbi.nlm.nih.gov/pubmed/17951842, 2007.

[9] E.E. Schadt, S.A. Monks, T.A. Drake, A.J. Lusis, N. Che, V. Colinayo, T.G. Ruff, S.B. Milligan, J.R. Lamb, G. Cavet, P.S. Linsley, M. Mao, R.B. Stoughton, and S.H. Friend, "Genetics of Gene Expression Surveyed in Maize, Mouse and Man," *Nature*, vol. 422, no. 6929, pp. 297-302, http://dx.doi.org/10.1038/nature01434, Mar. 2003.

[10] C. Huttenhower, A.I. Flamholz, J.N. Landis, S. Sahi, C.L. Myers, K.L. Olszewski, M.A. Hibbs, N.O. Siemers, O.G. Troyanskaya, and H.A. Coller, "Nearest Neighbor Networks: Clustering Expression Data Based on Gene Neighborhoods," *BMC Bioinformatics*, vol. 8, pp. 250-262, 2007.

[11] N. Song, J.M. Joseph, G.B. Davis, and D. Durand, "Sequence Similarity Network Reveals Common Ancestry of Multidomain Proteins," *PLoS Computational Biology*, vol. 4, no. 4, http://dx.doi.org/10.1371/journal.pcbi.1000063, Apr. 2008.

[12] R. Motwani and P. Raghavan, *Randomized Algorithms.* Cambridge Univ. Press, 1995.

[13] D. Gibson, R. Kumar, and A. Tomkins, "Discovering Large Dense Subgraphs in Massive Graphs," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB '05)*, pp. 721-732, 2005.

[14] R. Project, *The R Project for Statistical Computing*, www.r-project.org, 2003.

[15] M. Kanehisa, M. Araki, S. Goto, M. Hattori, M. Hirakawa, M. Itoh, T. Katayama, S. Kawashima, S. Okuda, T. Tokimatsu, and Y. Yamanishi, "Kegg for Linking Genomes to Life and the Environment," *Nucleic Acids Research*, gkm882+, vol. 36, pp. D480-484, http://dx.doi.org/10.1093/nar/gkm882, Dec. 2007.

[16] Ingenuity Systems "Ingenuity Systems Pathway Analysis," www.ingenuity.com, 2010.

[17] M. Ashburner, C.A. Ball, J.A. Blake, D. Botstein, H. Butler, J.M. Cherry, A.P. Davis, K. Dolinski, S.S. Dwight, J.T. Eppig, M.A. Harris, D.P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J.C. Matese, J.E. Richardson, M. Ringwald, G.M. Rubin, and G. Sherlock, "Gene Ontology: Tool for the Unification of Biology. The Gene Ontology Consortium," *Nature Genetics*, vol. 25, no. 1, pp. 25-29, http://dx.doi.org/10.1038/75556, May 2000.

[18] M.A. Bogue, S.C. Grubb, T.P. Maddatu, and C.J. Bult, "Mouse Phenome Database (mpd)," *Nucleic Acids Research*, vol. 35, pp. 643-649, http://dblp.uni-trier.de/db/journals/nar/nar35.html#BogueGMB07, 2007.

[19] Z. Yingyao, A. Young, S. Andrey, C. Kaisheng, Y.S. Frank, and A. Winzeler, "In Silico Gene Function Prediction Using Ontology-Based Pattern Identification," *Bioinformatics*, vol. 21, no. 7, pp. 1237-1245, http://dx.doi.org/10.1093/bioinformatics/bti111, Apr. 2005.

[20] E.H.A.M. Gordon and M. Regnier, "Regulation of Contraction in Striated Muscle," 2000.

[21] C.A. Conley,, K.L. Fritz-Six, A. Almenar-Queralt, and V.M. Fowler, "Leiomodins: Larger Members of the Tropomodulin (tmod) Gene Family," *Genomics*, vol. 73, no. 1, pp. 127-139, http://www.sciencedirect. com/science/article/B6WG1-45BCMW0-1/2/e5ef9707c be3d3959295dd41cf114e56, May 2001.

[22] K. Brix, P. Lemansky, and V. Herzog, "Evidence for Extracellularly Acting Cathepsins Mediating Thyroid Hormone Liberation in Thyroid Epithelial Cells," *Endocrinology*, vol. 137, no. 5, pp. 1963-1974, http://www.hubmed.org/display.cgi?uids=8612537, May 1996.
[23] L. Gautier, L. Cope, B.M. Bolstad, and R.A. Irizarry, "Affy—Analysis of Affymetrix Genechip Data at the Probe Level," *Bioinformatics*, vol. 20, no. 3, pp. 307-315, 2004.
[24] R.C. Gentleman, V.J. Carey, D.M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A.J. Rossini, G. Sawitzki, C. Smith, G. Smythy, L. Tierney, J.Y. Yang, and J. Zhang, "Bioconductor: Open Software Development for Computational Biology and Bioinformatics," *Genome Biology*, vol. 5, no. 10, pp. 80-95, http://dx.doi.org/10.1186/gb-2004-5-10-r80, 2004.

Banu Dost received the BS degree in computer science from Sabanci University, Istanbul, Turkey, in 2004. Currently, she is working toward the PhD degree in computer science and engineering at the University of California, San Diego. Her main research interests include bioinformatics and computational biology. She is particularly interested in developing graph-based optimization algorithms to analyze biological high-throughput data.

Chunlei Wu received the PhD degree in biomathematics and biostatistics from the University of Texas Graduate School of Biomedical Sciences in 2006. He is currently working as research investigator I in the Genomics Institute of the Novartis Research Foundation, San Diego, California. His main research interests include bioinformatics and computational biology. He is particularly interested in the analysis of high-throughput gene expression and expression QTL data.

Andrew Su received the bachelor's degree in chemistry and computer science from Northwestern University in 1998 and the PhD degree in chemistry from the Scripps Research Institute in 2002. He is currently serving as the group leader in bioinformatics and computational biology at the Genomics Institute of the Novartis Research Foundation. His research interests focus on data mining in high-throughput biology data and on creating web-based tools that apply the principle of community intelligence to biology.

Vineet Bafna is a professor at the University of California, San Diego. He has coauthored around 80 articles on diverse topics in bioinformatics, with a recent focus on algorithms for population genetics, genomics, and proteomics. He serves on the editorial board of *Biology Direct*, the *IEEE Transactions on Computational Biology*, and the *Journal of Algorithmic Biology*.