Published in final edited form as:

Methods Mol Biol. 2009 ; 553: 57–77. doi:10.1007/978-1-60327-563-7_4.

Manipulating Large-Scale *Arabidopsis* Microarray Expression Data: Identifying Dominant Expression Patterns and Biological Process Enrichment

David A. Orlando, Siobhan M. Brady, Jeremy D. Koch, José R. Dinneny, and Philip N. Benfey

Abstract

A series of large-scale *Arabidopsis thaliana* microarray expression experiments profiling genome-wide expression across different developmental stages, cell types, and environmental conditions have resulted in tremendous amounts of gene expression data. This gene expression is the output of complex transcriptional regulatory networks and provides a starting point for identifying the dominant transcriptional regulatory modules acting within the plant. Highly co-expressed groups of genes are likely to be regulated by similar transcription factors. Therefore, finding these co-expressed groups can reduce the dimensionality of complex expression data into a set of dominant transcriptional regulatory modules. Determining the biological significance of these patterns is an informatics challenge and has required the development of new methods. Using these new methods we can begin to understand the biological information contained within large-scale expression data sets.

Keywords

Clustering; microarray; gene expression; enrichment; gene ontology

1. Introduction

1.1. Microarray Technology, Large-Scale Data Sets, High-Resolution Data

A microarray is a rectangular slide with thousands of short stretches of DNA chemically bonded to it. The Affymetrix™ *Arabidopsis* ATH1 22 K microarray chip contains pieces of DNA, known as probes, corresponding to approximately 22,000 genes, and has become the quasi-standard in *Arabidopsis* expression profiling (1). Using this technology, two large data sets exist that have profiled expression at two different scales. The AtGenExpress data set contains three series of expression profiles, with each series examining a different set of cells or conditions. The developmental timeline series contains profiles of expression in a different organ or developmental stage (2). The hormone series contains profiles of

© Humana Press, a part of Springer Science+Business Media, LLC 2009

¹⁹In essence, when we consider enrichment of biological features, we are testing the enrichment of each feature at a time, and thus, we are testing multiple hypotheses. False discovery rate (FDR) is a statistical method used in multiple hypotheses testing to correct for multiple comparisons (18). The FDR controls the expected proportion of incorrectly rejected null hypotheses (type I errors) (23). The *Q*-value of a test measures the proportion of false positives incurred (the FDR) when that particular test is called significant.

expression in seedlings in response to various hormones over time (3). Finally, the abiotic and biotic stress series contains profiles of expression from seedlings in response to biotic stimuli like pathogens and to environmental abiotic stimuli (4). In total, the AtGenExpress data set profiles the expression of approximately 22,000 genes in roots, shoots, seedlings or cell culture under many conditions. A second large data set contains fewer experiments, but contains expression profiles at much higher resolution than the AtGenExpress data set, with 19 different experiments profiling expression of nearly all cell types within the *Arabidopsis* root, in addition to 13 developmental time points (5–7). Together, these data sets contain massive amounts of expression information; however, extracting biological insights from this data is a real challenge. A useful approach is to look for groups of genes that are expressed in similar patterns across the different conditions. Expression similarity often implies co-regulation and can be used to identify transcriptional regulatory modules. Thus finding groups of genes with common patterns of expression is a good starting point for extracting biological insight from large expression data sets.

1.2. Identifying Dominant Expression Patterns and Associated Genes

The informatic task of grouping genes with similar expression patterns is commonly referred to as clustering. Clustering can be more rigorously defined as the task of separating a large set of elements (genes) into distinct subsets (groups/clusters of genes) such that all elements in a subset share a common feature (similar expression pattern). The similarity between two elements is defined by a distance metric, such as Euclidean distance or Pearson correlation. There are a wide variety of clustering algorithms which use different strategies to separate the full set of elements into subsets, with the most common being the hierarchical and K-means algorithms.

There are advantages and disadvantages to using one clustering algorithm over another. The K-means algorithm is useful for finding subsets with many members, but can force elements into subsets where they may not belong simply because all the other subsets are worse matches. Furthermore, the K-means algorithm begins with randomly chosen cluster centers, and different runs of the algorithm can result in subsets containing different members. Finally, K-means clustering will split the full set of elements into exactly K subsets, which is user-determined and may not be optimal. In contrast, hierarchical methods are good at identifying relationships between single elements or small groups but it is often difficult to determine the proper subsets as the set of elements grows large.

We present a method which utilizes both of these algorithms in an effort to identify a set of unique dominant expression patterns within a large gene expression data set. We define a “dominant expression pattern” to be a pattern that has strong support (i.e., many genes exhibit the pattern) within the data. Thus our method does not try to necessarily try to find all the different expression patterns present in a data set, only those which represent large groups of co-expressed genes. The method utilizes a variant of the K-means algorithm, fuzzy K-means (8), to create a preliminary set of groups, from which initial patterns can be defined. Using a K-means-based method ensures that the initial set of patterns will be “dominant”, since the K-means algorithm will preferentially create subsets with many members. We then use hierarchical clustering, and its strengths in identifying relationships

between a small number of elements, to cluster and collapse initial patterns that are similar to each other, resulting in a final set of unique dominant expression patterns. Once these unique dominant expression patterns have been identified, clusters of genes associated with each of the patterns can be assembled and analyzed for biological significance.

1.3. Analyzing Clusters of Genes for Biological Significance: Biological Information Resources

Once a cluster has been obtained, a second challenge is to determine the biological significance, if any, of this group of co-expressed genes. The “biological significance” of a gene or a group of genes can be subjective. It could mean the biochemical, developmental, or physiological process that the gene’s product is involved in. Alternatively, if one is interested in the role of a gene in the context of transcriptional regulatory modules, which is particularly important when monitoring the output of the modules using microarray analysis, then biologically significant information could include *cis*-regulatory elements present within the upstream or downstream regulatory regions of the gene. If the gene is a transcription factor, it could be useful to know its transcription factor class and its binding site preference. Literature is often a good source for mining this type of information, and several consortia are responsible for mining literature for experiments that identify these biological features for individual genes and then storing these annotations in a database. The most popular of these consortiums is the Gene Ontology or GO consortium. Gene Ontology categories are controlled vocabulary terms that describe the biological process, molecular function, or cellular component that is associated with a gene in many model organisms (9). The Arabidopsis Information Resource (TAIR) curates the literature and maintains the *Arabidopsis* GO category database (10). Several databases exist that define and catalog *Arabidopsis* transcription factors as well as identified *cis*-regulatory elements. These databases include the database of Arabidopsis transcription factors (DATF), PLACE (plant *cis*-acting DNA regulatory elements), and the Arabidopsis gene regulatory information server (AGRIS) (11–13). Microarray expression analysis has become a useful tool for experimenters performing *Arabidopsis* research, and many publications exist that use microarrays to identify genes associated with particular pathways. For example, some of these studies have identified genes associated with the M-phase or S-phase of the cell cycle, genes expressed during root hair morphogenesis, or genes associated with primary or secondary cell wall biosynthesis (14–17). These studies often identify associated genes using rigorous statistical testing; however, TAIR and other resources do not currently compile genes associated with biological processes via microarray analysis in their databases. We have also mined the literature for these types of studies and have included them as a source of useful biological information to infer biological significance (7).

1.3.1. Testing for Enrichment: Multiple Hypothesis Testing—These consortia and statistical resources are rich in information. When given a set of genes exhibiting a similar expression pattern one can search this set for obvious relationships among the genes. For example, if one finds that several genes that are known to act in a particular biochemical pathway are coordinately expressed, it suggests that this biochemical pathway is of particular importance in the cell-type, developmental stage, or environmental condition experimentally tested. In essence, one can test for the enrichment of a biological feature

within a data set. However, given the vast number of potential biological features, the statistical significance of enrichment when testing each individual hypothesis is of utmost importance. A common statistical method used to determine the significance of feature enrichment is the hypergeometric distribution (18). The hypergeometric distribution tests whether any feature or variable is found in a list at a frequency greater than would be expected by chance and calculates a *P*-value. This method has been used in particular to identify GO term enrichment and *cis*-regulatory enrichment in large-scale gene expression analysis (19, 20). See Note 17 for alternative methods to determine the significance of these biological features.

1.3.2. Choice of Correct Background Distribution of GO Categories—

Enrichment testing entails determining the significance of a number of categories present within a list of interest relative to the number of times these categories are found in a background list. Therefore the choice of correct background is imperative. Current web-based methods exist that test for enrichment of GO categories, including the Generic GO Term Finder and ATHENA (19, 20). These methods consider the background as all genes annotated with a GO category. In the context of microarray analysis, however, we must consider that our background is the set of genes being tested for enrichment, that is, all genes present on the microarray being used. Not all genes annotated in the *Arabidopsis* genome are present on the ATH1 22 K microarray. We have therefore developed a method that tests enrichment relative to only the genes whose expression we are testing (i.e., those present on the ATH1 microarray chip).

2. Materials

1. **R Software:** R is a free software environment designed for statistical computing and is available for download from <http://www.r-project.org/>(21). The code for generating the set of unique dominant patterns is written for R.
 2. **R Scripts** (Table 4.1)
 - a. Code for generating initial set of clusters.
 - b. Code for identifying unique dominant expression patterns and their associated genes.
 - c. <<http://www.arexdb.org/software.jsp>>
 3. **Background Chip: ATH1 Chip:** Tab-delimited text file containing all Affymetrix™ probe sets in one column and all corresponding AGI chromosome locus identifiers in a second column.
- Singleton Chip:* Tab-delimited text file containing all Affymetrix™ probe sets which map to only a single AGI chromosome locus identifier in one column and corresponding AGI locus identifiers in a second column.

¹⁷TAIR periodically releases new annotations of the *Arabidopsis* genome (10). With each new version of the *Arabidopsis* genome, a new file describing the Affymetrix™ probe set to AGI locus identifier conversion is also released. We recommend using the most recent version of this release, although it is important to always use the same version for all rounds of data analysis when considering enrichment in multiple gene lists.

4. *Biological Information Files*

- a. **GO Annotation File:** Tab-delimited text file containing AGI locus identifiers in the first column, corresponding gene models in a second column, GO category descriptions in a third column, and GO IDs in a fourth column.
 - b. **Array Annotation File:** Tab-delimited text file containing AGI locus identifiers in one column, and biological processes annotated to these genes as determined by mining the literature in a second column.
 - c. **Transcription Factor Family File:** Tab-delimited text file containing transcription factor AGI locus identifiers in one column and the family that these transcription factors belong to in a second column. The transcription factor family file was created by querying three transcription factor databases: DATF, AGRIS, and Riken (7, 11, 13, 22). If a transcription factor was annotated as belonging to a particular family in two of three databases, it was included in this list.
 - d. **Query List:** Tab-delimited text file containing the AGI locus identifiers in the first column. This is the list that will be tested for biological enrichment relative to the background.
5. Hypergeometric Distribution: <http://jakarta.apache.org/commons/math>
 6. **Q-Value Test:** John Storey's Q-value Method (23): <http://faculty.washington.edu/jstorey/qvalue/>
 7. **Biological Enrichment Software:** <http://www.arexdb.org/software>

3. Methods

3.1. Identifying Unique Dominant Expression Patterns

3.1.1. Pre-clustering Data Filtering—Filtering input expression data is an important step in recovering dominant expression patterns of interest (*see* Note 1 for expected input data format). Is it often the case that when an expression data set profiles only a subset of tissues or organs (i.e., the root is a subset of the whole plant) there will be many genes which are not expressed above a reasonable cutoff in any measurement in the subsystem being analyzed? When looking for unique dominant expression patterns, these genes should be removed. Additionally, there is often another subset of genes that are expressed uniformly across all the measurements. These genes are also not useful in identifying novel dominant expression patterns and should be removed (*see* Note 2 for more information about flat expression patterns). Filtering the expression data is useful for three reasons: (1) it removes genes that are not informative in identifying dominant expression patterns, (2) genes with relatively flat expression profiles (i.e., all-off or non-varying) can behave poorly when compared using the Pearson correlation distance metric (*see* Note 3) and it reduces the size

¹The input data should be a tab-delimited text file with each row being a gene and each column being a numeric measurement. The first column should contain a unique string identifying the gene in each row. The first row should contain unique strings identifying the measurements (tissues, developmental zones, etc.). Missing values should be reported as "NA".

of the input to the clustering step which will decrease runtime of the fuzzy K-means algorithm. The rigor of this filtering can be adjusted in the code provided to suit the user's needs. Finally, if the input data set has already been filtered such that all the genes in the input data display some characteristic of interest, these filtering steps do not need to be applied.

3.1.2. Data Clustering

3.1.2.1. Normalization via $\log_2$ Transformation: When using non-fold normalized data (i.e., one-color Affymetrix™ arrays), it is important to normalize the data by a $\log_2$ transformation after filtering (*see* Note 5 for definition of $\log_2$ normalization). This ensures that individual genes are compared on the basis of the shape of their expression patterns and not the absolute levels (*see* Note 4 regarding comparing absolute levels). Two-color spotted arrays are usually measured in the form of a fold ratio between the observation and a control, and thus usually do not need to be $\log_2$ normalized.

3.1.2.2. Choice of Distance Metric: The code provided in Table 4.1 calculates the similarity between expression profiles via Pearson correlation (*see* Note 6 for the relationship between distance and similarity measures). Pearson correlation is used because it is sensitive to the shape of the expression profile while being relatively amplitude

²The method has the ability to remove genes that are not varying across the measurements, either because they are constitutively on or off. As stated in Section 3.1.1, the reason for removing these genes is 3-fold: (1) there can be a very large number of these flat profiles that can obscure other patterns when performing the initial clustering, (2) certain distance metrics, such as Pearson correlation can behave erratically when comparing flat profiles (*see* Note 4), and (3) it reduces the size of the input to the clustering step which will decrease runtime. Additionally, if the user feels that the flat pattern is important, particularly in the cases where genes are ubiquitously expressed, it is easy to manually add it to the set of patterns and thus does not need to be computationally modified (*see* Note 11). Even if the user decides not to add a flat pattern into the final set, identifying genes which are expressed ubiquitously in all measurements can be biologically informative and could be another useful analysis. These genes can be identified by selecting genes with low variance and which are expressed above a given cutoff in every measurement.

³Pearson correlation is a common measure of similarity between two genes. To a first approximation, two genes are well correlated if their expression patterns have the same overall shape across multiple experiments, even if the amplitude changes of the individual curves are different. As an example, two genes will be well correlated if they are both upregulated, even if one is upregulated 10-fold and the other is only 2-fold. However, this amplitude insensitivity is problematic when one of the genes has a profile that is nearly flat. A gene with a flat profile often has small unimportant up/down movements simply due to measurement noise. The problem arises from the fact that you can get large positive (or negative) Pearson correlations between noisy flat genes and other genes when the tiny random up/down movement in the noise happens to coincide with the biologically significant up/down movement of the other gene. In short, Pearson correlation is a very useful measure, but one should be careful when using it with genes whose expression changes are mostly due to measurement noise.

⁵In order to make the genes with expression patterns of very different amplitudes easily comparable, it is suggested that you $\log_2$ transform the expression data. The $\log_2$ normalization of a gene is done by dividing each of the gene's individual measurements by its mean value across all measurements, and then taking the log base 2 of that number. After the normalization, a value of 0 corresponds to a original measurement that was exactly the mean expression, a value of 1 corresponds to a original value that was twice or the mean expression, and a value of -2 corresponds to a original value that was 4-fold lower than the mean. This normalization is useful because it places all genes on the same measurement scale, even if they are expressed at very different levels in the cell.

⁴Some microarray platforms, such as the Affymetrix™ platform, will report gene expression in terms of absolute units as opposed to the fold change relative to a control which is reported for spotted two-color arrays. These absolute measurements are very useful as they can reveal the different biological concentrations of mRNA transcripts in the cell. However, the basal concentrations of two genes can be very different even though the genes are part of the same process. 72 Orlando et al. This can make the co-expression of the two genes difficult to detect because of this difference in scale. Normalization using $\log_2$ transformation can help alleviate this problem.

⁶Most clustering algorithms require a notion of distance, or dissimilarity, between two objects, where a larger number denotes larger distance or dissimilarity. The code provided uses Pearson correlation, which is a measure of similarity, and thus the method needs to convert from the similarity measure to distance. It does this by subtracting the Pearson correlation value from 1. This means that a high Pearson correlation value (1 ¼ perfect correlation) will be converted to 0 (small distance) and a low Pearson correlation value (-1 ¼ perfect anticorrelation) to 2 (high distance). The method then divides this number by 2, simply to scale the distances between 0 and 1. One needs to be careful when using similarity measures in conjunction with distance measures, to ensure that the interpretation of large and small values is consistent.

insensitive. Using other distance metrics is possible, but requires modification of the code (see Note 7).

3.1.2.3. Fuzzy K-means Clustering: The clustering method used in the provided code (Table 4.1) is an implementation of fuzzy K-means algorithm described in Chapter 4 of Kaufmann and Rousseeuw (8). The exact method is named “FANNY” and is implemented in the “cluster” package in R (24). The implementation, as it is used in the code, takes five parameters as input: *x*, *k*, *diss*, *memb.exp*, and *maxit*. The *x* variable contains the distance of each gene from every other gene. The *k* variable is the number of initial clusters desired (see Note 8). The *diss* variable should always be set to TRUE, indicating that the *x* variable contains distances. The *memb.exp* and *maxit* variables set the membership exponent and maximum iterations used by the implementation. Details on all the parameters can be found in the help files contained within the cluster R package, and can be accessed by typing “? fanny” in the R environment. The FANNY implementation returns a large object, with many components, all of which are detailed in the package help files. Our method uses only the output contained within the membership component of the returned object. The membership component is a matrix with one row for each gene and one column for each of the K clusters. The value in each cell is the probability that the gene belongs to a given cluster. We use this probability information to determine which genes belong to multiple clusters and which genes do not match well to any cluster.

3.1.3. Creating the Initial Patterns—The initial set of patterns is generated by taking the column-wise median of the expression profiles for all genes truly belonging to each cluster. The notion of truly belonging to a cluster is where the probability information in the membership component of the FANNY output is incorporated. Our method uses a probability cutoff variable, *clusterCutoff*, which is used to determine which genes belong to which cluster. A gene belongs to a cluster if the membership component of the fuzzy K-means clustering reports a probability of that gene being in that cluster at or above the value in the *clusterCutoff* variable. The code provided (Table 4.1) allows a user to set this cutoff themselves (*autoSelectClusterCutoff* ¼ FALSE) or determine it automatically (*autoSelectClusterCutoff* ¼ TRUE). If the *clusterCutoff* is determined automatically, the method finds a cutoff at which the average gene belongs to one cluster. Often, the cutoff will be below 0.5, allowing for some genes to truly belong to multiple clusters, and thus be included in the creation of multiple initial patterns. If another clustering method were used, the initial set of patterns could simply be the mean expression profile of all the genes

⁷The code provided uses Pearson correlation as its distance metric. Other distance metrics can be substituted in its place, by changing the code at the appropriate places. Other distance metrics may be appropriate if one wants to define similarity not on the basis of correlation, but rather on the basis of absolute level (Euclidean distance would be appropriate), or other criteria for which a different measure is more suitable. When choosing a distance metric, be sure to consider whether or not the data should be normalized by log₂ transformation. It is also important to consider how the patterns should be created and how similar patterns are collapsed (the column-wise median is usually appropriate, but may not be for some specialized applications).

⁸The choice of K when using a K-means algorithm is very important. Although our method seeks to mitigate the impact of the choice by collapsing similar patterns, the Pattern Identification and Biological Process Enrichment 73 choice of K should still be considered carefully. Even though there are methods such as the gap statistic (26) which can be helpful in suggesting a good K, finding the correct choice of K can often be more of an art than a science. We suggest you start with a relatively high K initially. A good rule of thumb is to use a K 25% higher than the number of patterns you expect to find. It is also useful to run the method with higher and lower choices of K and compare the final sets of patterns.

assigned to each cluster by the algorithm (*see* Note 9 for information on how to use a different clustering algorithm).

3.1.4. Collapsing Similar Patterns—The final step in creating the set of unique dominant expression patterns is to collapse similar patterns created in the step above. Collapsing similar patterns corrects for the tendency of the K-means clustering algorithm to split large, potentially similarly expressed groups of genes and ensures that one has obtained the full complement of distinct, dominant expression patterns. The distances between each of the patterns are computed using the same distance metric used in the initial clustering (Pearson correlation in the code provided). These distances are then used to hierarchically cluster the initial set of patterns. The resulting tree is cut at a user-defined cutoff (stored in the *pattern Similarity Cutoff* variable), and the patterns within the resulting clusters are collapsed together. In this method, we have used single-linkage clustering as the default. Using single-linkage hierarchical clustering ensures that all patterns whose distance is less than the cutoff are collapsed together (*see* Note 10). The default value in the code is *pattern Similarity Cutoff* $\frac{1}{4}$ 0.1, which ensures that all patterns with a Pearson correlation at or above 0.9 (or a Pearson correlation “distance” of 0.1) are collapsed together. When multiple patterns are collapsed the new pattern is the column-wise median of all the collapsed patterns. After all the patterns are collapsed, the resulting rows are a set of unique dominant expression patterns.

3.2. Assignment of Genes to Clusters

Given this set of unique dominant expression patterns, is it useful to have lists of genes which exhibit each respective expression pattern? *See* Note 11 for how to add additional user-defined patterns before gene assignment. Using the same distance measure employed to identify the dominant expression patterns, our method calculates the distance from each gene to each pattern. The code provided only examines those genes that were used in K-means clustering algorithm as genes of interest, but this can be modified (*see* Note 12). Given the calculated distances, the groups of genes assigned to a pattern are defined as the set of genes with a distance below a certain cutoff (or above a certain cutoff if measuring using a similarity measure like Pearson correlation). The default value in the code is *pearsonCutoff* $\frac{1}{4}$ 0.85, which associates a gene with a pattern if the gene Pearson correlates

⁹The code provided uses a fuzzy K-means algorithm and its associated output to determine which genes should be used to build the initial patterns. The advantage of the fuzzy K-means algorithm is that the output can be used to detect genes that should be used in multiple initial patterns and genes that should not be used at all. You can use any clustering algorithm of your choice and rely on the hard cluster assignments to build the initial set of patterns. You will need to modify the code (in particular the *getClustProfiles* subroutine) to take the output of your clustering method and correctly build the initial set of patterns.

¹⁰Linkage refers to the way distance between two clusters of elements is calculated. There are three common types of linkage used in hierarchical clustering: single, complete, and average. The distance between two clusters using single linkage is the minimum distance between any element in cluster 1 and any element in cluster 2. For complete and average linkage, the distance between two clusters is the maximum and mean distances between any element in cluster 1 to any element in cluster 2, respectively. Single linkage is used in the code provided.

¹¹If the user wishes, they can import an additional file of expression patterns that will be added to the final set of patterns. The user can define these patterns in a tab-delimited text file in the same format as the input data (*see* Note 1) except the first column will contain a unique pattern name. The user can then include them by setting the user *Pattern-File* parameter to the name of that file. The method assumes the data in this file are in the same format and scale as the other patterns (pre-log₂ transformed is necessary) and that the column order is the same. For adding in a flat pattern, the expression values could be set to all zeros. The user should be careful about adding in multiple additional patterns, because these patterns will not be collapsed. If the user chooses a distance metric which is sensitive to amplitude, then precise choice of values in the pattern file should be considered carefully. 74 Orlando et al.

to the pattern at or above 0.85. The advantage of doing assignment in this manner is that it allows genes to belong to more than one pattern if it is similar to multiple patterns, or, not be assigned to any pattern (25). The method produces a table with a set of genes assigned to each pattern. Having the complete set of genes matching an expression pattern is very desirable, especially when using this set to look for biological process enrichment. *See Note 13 for details of the method output formats.*

3.3. Biological Process Enrichment

3.3.1. Process Background Chip—We utilize two types of background sets in our analysis. The first is the “ATH1 chip” and the second is the “Singleton Chip”. (*See Section 2.3 and Note 15 for additional details regarding the differences between these two backgrounds. See Note 16 for information regarding nuclear, chloroplast- and mitochondrial-encoded genes.*) This file contains a list of Affymetrix™ probe set identifiers in the first column and a list of their corresponding AGI locus identifier in a second column (*see Note 17 for information regarding version releases of these conversions*).

3.3.1.1. Singleton Chip Preprocessing: The singleton chip is first processed by removing Affymetrix™ probe set identifiers that are mapped to multiple AGI locus identifiers. In the first column any rows with Affymetrix™ probe set identifiers that contain `<_x_at>` or `<_s_at>` are removed (these identifiers indicate that a probe set matches to multiple loci). In the second column, any rows with multiple AGI locus identifiers `<;>` are also removed, in cases where `<_x_at>` or `<_s_at>` notation was not indicated (*see Note 15*).

3.3.1.2. Background Chip Reverse Mapping: Since we are testing for enrichment of features that are annotated to AGI locus identifiers, we must obtain a count of all AGI chromosomal loci found on the microarray chip. For ease of counting, the first and second

¹²The code currently calculates the correlation between each gene in the *expDataFiltered* variable and each pattern. The *expDataFiltered* variable contains all the genes and their expression which was used in the fuzzy K-means clustering. To associate a different set of genes, simply pass a different matrix of expression to the *correlateGenesToProfiles* subroutine. One common change would be to replace the line in the code:

```
“geneToPatternCor<- correlateGenesToProfiles(expDataFiltered, finalProfiles)” with
```

```
“geneToPatternCor<- correlateGenesToProfiles(expressionData, finalProfiles)”.
```

That would calculate the association between all genes in the input data to all the patterns. If one wishes to use a different subset, modify the code so that the subroutine is passed a matrix (in the same format as *expressionData*) containing only genes of interest.

¹³The method will create three files. One file is an .rDump file (*methodResultFile*), which is a file that can be read into R using the *load()* command. This contains the information needed for and produced by the method and are not human readable. The method also produces a tab-delimited text file (whose filename is user specified with the *patternOutputFile* parameter), which has the same general format as the input expression data, but each row contains the expression for a dominant expression pattern, as opposed to a single gene. Finally, the method also produces a tab-delimited text file (whose filename is user specified with the *groupOutputFile* parameter) which defines the sets of genes associated with each pattern. Each column corresponds to a different pattern (with the pattern name defined in the first row), and each row contains the name of a gene which is associated with that pattern.

¹⁵As probe sets annotated to multiple genes can obscure the biological relevance of enrichment, in the majority of cases, we choose to use the “Singleton Chip”.

¹⁶In our analysis we have chosen to only consider nuclear-encoded genes. However, the ATH1 chip file does contain probe sets which map to mitochondrial- or chloroplast-encoded genes, and the GO annotation file which is available for download from TAIR also contains ontologies associated with mitochondrial- and chloroplast-encoded genes. We therefore remove these genes from the ATH1 chip file (and indirectly the singleton chip file), and from the GO annotation file. If one chooses to consider also enrichment among these mitochondrial- and chloroplast-encoded genes, one must include these in both the ATH1 chip file and the GO annotation file for proper comparison.

columns are reversed. The number of AGI locus identifiers in the first column is then counted and stored for use in the hypergeometric distribution test.

3.3.2. Process Gene Descriptor Map (GO Annotations/Array Annotations/TF family)

3.3.2.1. GO Annotation File Preprocessing: The GO annotation file requires additional processing relative to the array annotation file and transcription factor family file. The GO annotation file contains four columns: <AGI ID>, <model>, <description>, and <GO ID>. The second column, <model>, is disregarded in our analysis (*see* Note 14 for additional details).

3.3.2.2. Gene Descriptor Map: As mentioned in Introduction, we only test for the enrichment of features relative to the appropriate background. We therefore filter the GO annotation list, array annotation, and transcription factor family lists so that they only contain the AGI locus identifiers present on the selected background chip. These filtered files are now named the <gene descriptor map> file. In the case of GO categories and array annotations, an AGI locus identifier can have multiple terms annotated to it; therefore, an AGI locus identifier can appear multiple times in the left column.

3.3.3. Process Enrichment Relative to Gene Descriptor Map—The query list is read in to the program. If multiple AGI locus identifiers are found in a single row, these locus identifiers are removed, as they were most likely obtained from an <_s_at> or an <_x_at> Affymetrix™ probe set to AGI locus identifier conversion, and will bias the biological interpretation of results. The gene descriptor map is now also read into the program. A set of commands determines the measurements needed for input into the hypergeometric distribution test. For all further sections the term GO ID is interchangeable with <array annotation> or <TF family>. These methods can also be used for enrichment of any other biological features, including the enrichment of *cis*-elements.

Measurements needed for input into the hypergeometric distribution set:

- A. The subset of AGI locus identifiers that are in common between these two files, and their associated GO IDs are filtered. For GO ID x, the number of AGI locus identifiers associated with it in the query list is counted. This is re-iterated for each GO ID present in the query list.
- B. For GO ID x, the number of AGI locus identifiers associated with it in the background chip.

¹⁴ We download the GO annotation file from TAIR and modify it into the user format described. First, as described in Note 16, we only include nuclear-encoded genes. We then create, for each chromosome a list of AGI locus identifiers, the corresponding gene models, GO description, and GO ID information as described in Section 2.4.1. When the GO IDs are associated with corresponding AGI locus identifiers (*see* Section 3.3.3), only unique AGI locus identifiers are considered in our counts for the hypergeometric distribution test. A number of gene models can exist for each AGI locus identifier (gene). These gene models can describe the protein-coding sequence, noncoding molecular species, or alternately spliced variants (10) associated with each gene. Although we ignore these models when we use this GO annotation file (*see* Section 3.3.2.1), after obtaining a statistically significant feature using the hypergeometric distribution test, one should cross-reference Pattern Identification and Biological Process Enrichment 75 the gene model sequence associated with each significant feature, with the corresponding probe sequences on the ATH1 22 K microarray chip to determine which gene model is most appropriate for further consideration.

- C. The number of AGI locus identifiers present on the query list.
- D. The number of AGI locus identifiers present on the background chip.

3.3.4. Hypergeometric Distribution Test—These four numbers are provided to the hypergeometric distribution test script (*see* Section 2.7). The resulting *P*-values are then used to assess the false discovery rate (FDR) (*see* Note 18). We choose to accept a significant enrichment of $P = 10^{-3}$, although use of the FDR as a threshold is also recommended. Since we only consider enrichment in our query list, we only consider lists where the percentage of the query GO count/query set size is >background count/background set size.

3.3.5. Q-Value Testing—To assess the FDR we use John Storey's QVALUE program (23). This program runs in the R software environment. *See* Section 2.1 for the direct link to this software. *P*-values must be stored in a tab-delimited text format to be read into QVALUE.

References

1. Busch W, Lohmann JU. Profiling a plant: expression analysis in *Arabidopsis*. *Current Opinion in Plant Biology*. 2007; 10(2):136–141. [PubMed: 17291825]
2. Schmid M, Davison TS, Henz SR, et al. A gene expression map of *Arabidopsis thaliana* development. *Nature Genetics*. 2005; 37(5):501–506. [PubMed: 15806101]
3. Nemhauser JL, Hong F, Chory J. Different plant hormones regulate similar processes through largely nonoverlapping transcriptional responses. *Cell*. 2006; 126(3):467–475. [PubMed: 16901781]
4. Kilian J, Whitehead D, Horak J, et al. The AtGenExpress global stress expression data set: protocols, evaluation and model data analysis of UV-B light, drought and cold stress responses. *Plant Journal*. 2007; 50(2):347–363. [PubMed: 17376166]
5. Birnbaum K, Jung JW, Wang JY, et al. Cell type-specific expression profiling in plants via cell sorting of protoplasts from fluorescent reporter lines. *Nature Methods*. 2005; 2(8):615–619. [PubMed: 16170893]
6. Birnbaum K, Shasha DE, Wang JY, et al. A gene expression map of the *Arabidopsis* root. *Science*. 2003; 302(5652):1956–1960. [PubMed: 14671301]
7. Brady SM, Orlando DA, Lee J-Y, et al. A high-resolution root spatiotemporal map reveals dominant expression patterns. *Science*. 2007; 318(5851):801–806. [PubMed: 17975066]
8. Kaufman, L.; Rousseeuw, PJ. *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: Wiley; 1990.
9. Ashburner M, Ball CA, Blake JA, et al. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics*. 2000; 25:25–29. [PubMed: 10802651]
10. Swarbreck D, Wilks C, Lamesch P, et al. The Arabidopsis Information Resource (TAIR): gene structure and function annotation. *Nucleic Acids Research*. 2007 gkm965.
11. Guo A, He K, Liu D, et al. DATF: a database of *Arabidopsis* transcription factors. *Bioinformatics*. 2005; 21(10):2568–2569. [PubMed: 15731212]
12. Higo K, Ugawa Y, Iwamoto M, Korenaga T. Plant cis-acting regulatory DNA elements (PLACE) database. *Nucleic Acids Research*. 1999; 27(1):297–300. [PubMed: 9847208]
13. Palaniswamy SK, James S, Sun H, Lamb RS, Davuluri RV, Grotewold E. AGRIS and AtRegNet. A platform to link cis-regulatory elements and transcription factors into regulatory networks. *Plant Physiology*. 2006; 140(3):818–829. [PubMed: 16524982]

¹⁸. Alternate methods used to identify significance of enriched features are described in (27).

14. Brown DM, Zeef LAH, Ellis J, Goodacre R, Turner SR. Identification of novel genes in *Arabidopsis* involved in secondary cell wall formation using expression profiling and reverse genetics. *Plant Cell*. 2005; 17(8):2281–2295. [PubMed: 15980264]
15. Jones MA, Raymond MJ, Smirnov N. Analysis of the root-hair morphogenesis transcriptome reveals the molecular identity of six genes with roles in root-hair development in *Arabidopsis*. *Plant Journal*. 2006; 45(1):83–100. [PubMed: 16367956]
16. Menges M, de Jager SM, Gruijssem W, Murray JAH. Global analysis of the core cell cycle regulators of *Arabidopsis* identifies novel genes, reveals multiple and highly specific profiles of expression and provides a coherent model for plant cell cycle control. *Plant Journal*. 2005; 41(4): 546–566. [PubMed: 15686519]
17. Persson S, Wei H, Milne J, Page GP, Somerville CR. Identification of genes required for cellulose synthesis by regression analysis of public microarray data sets. *Proceedings of the National Academy of Sciences of the United States of America*. 2005; 102(24):8633–8638. [PubMed: 15932943]
18. Gadbury GL, Garrett KA, Allison DB. Challenges and approaches to statistical design and inference in high dimensional investigations. In this volume.
19. Boyle EI, Weng S, Gollub J, et al. GO::TermFinder – open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes. *Bioinformatics*. 2004; 18:3710–3715. [PubMed: 15297299]
20. O'Connor TR, Dyreson C, Wyrick JJ. Athena: a resource for rapid visualization and systematic analysis of *Arabidopsis* promoter sequences. *Bioinformatics*. 2005; 24:4411–4413. [PubMed: 16223790]
21. Team, RDC. R: A Language and Environment for Statistical Computing. Vienna: R Foundation for Statistical Computing; 2006.
22. Iida K, Seki M, Sakurai T, et al. RARTF: database and tools for complete sets of *Arabidopsis* transcription factors. *DNA Research*. 2005; 12:247–256. [PubMed: 16769687]
23. Storey JD. A direct approach to false discovery rates. *Journal of the Royal Statistical Society, Series B*. 2002; 64:479–498.
24. Maechler, M.; Rousseeuw, PJ.; Hubert, M.; Hornik, K. Cluster: Cluster Analysis Basics and Extensions. 9 ed.. R package version; 2007. 1.11.
25. Gasch A, Eisen M. Exploring the conditional coregulation of yeast gene expression through fuzzy k-means clustering. *Genome Biology*. 2002; 3(11) research0059.1–research 22.
26. Tibshirani, R.; Walther, G.; Hastie, T. Technical Report 208. Department of Statistics, Stanford University; 2000. Estimating the number of clusters in a dataset via the gap statistic.
27. Levine DM, Haynor DR, Castle JC, et al. Pathway and gene-set activation measurement from mRNA expression data: the tissue distribution of human pathways. *Genome Biology*. 2006; 7(10):R93. [PubMed: 17044931]

Table 4.1

Documentation of, and R source code for, software for clustering, pattern identification and gene assignment. Each portion of the table documents a particular section of the software and includes a short text description, with important features and considerations highlighted, and the relevant R source code (within the black boxes). The sections are presented in the order they should be run during an analysis. The source code can be downloaded from <http://www.arexdb.org/software>.

Code for Fuzzy K-Means clustering		
User specified parameters		
These seven parameters need to be set before each run of the method.		
Parameter	Type	Description
inputFile	string	The name of the file with the tab-delimited expression data. See note 4.1.1 for required file format.
minExpFilter	decimal	This parameter is the minimum expression a gene must have in order to pass the low expression filter. Setting this to FALSE will skip the low expression filtering.
minVarFilter	integer	This parameter controls what percentage of genes, ranked by variance, are removed. A value of 75 will filter out the bottom 75% of the genes (thus retaining the top 25%). Setting this to FALSE will skip the low variance filter.
kChoice	integer	This integer sets the initial number of patterns found by the fuzzy K-means algorithm. See note 4.1.8.
fuzzyKmemb	decimal	This sets the membership exponent parameter in the fuzzy K-means algorithm. Read the <code>cluster</code> package documentation before adjusting this.
alreadyLog2	TRUE/FALSE	This flag tells the method if the data needs to be log2-fold transformed before computing the distance between each gene. Set this to TRUE if the input data is already in terms of fold change. See note 4.1.5.
methodResultFile	string	The name of the <code>rData</code> file where the results of the clustering will be stored.
diagnosticFile	string	The name of the output file where diagnostic and status information will be printed.
<pre>inputFile <- "inputData.txt" minExpFilter <- FALSE minVarFilter <- 50 kChoice <- 15 fuzzyKmemb <- 1.05 alreadyLog2 <- FALSE methodResultFile <- "patternIdent_result.rDump" diagnosticFile <- "clustering_diagnostic.txt"</pre>		
Reading the input data		
This code loads the required cluster package and reads the expression data. It also creates some book-keeping variables.		
<pre>library(cluster)</pre>		

Code for Fuzzy K-Means clustering

User specified parameters

These seven parameters need to be set before each run of the method.

Parameter	Type	Description
<pre>removeLowE <- 0 removeLowV <- 0 dateRun<- date() cat("Starting Fuzzy K-Means clustering on ",dateRun,"\n",file=diagnosticFile) expressionData <-read.table(file=inputFile,sep="\t",header=TRUE) rownames(expressionData)<-expressionData[,1] expressionData <- expressionData[,2:ncol(expressionData)] cat("Expression data read from ",inputFile,"\n",file=diagnosticFile,append=TRUE) cat("\t",nrow(expressionData)," genes with ",ncol(expressionData)," observations. \n",file=diagnosticFile,append=TRUE) expDataFiltered <- expressionData filterSettings <- paste("Filter Settings:\n\tInput File = ",inputFile,sep="")</pre>		

Running low expression filter

This is the code for the low expression filtering. If minExpFilter is not set to FALSE it will remove any genes which are not expressed above minExpFilter in any measurement (column).

```
cat("Removing low expressed genes - ",file=diagnosticFile,append=TRUE)
filterSettings <- paste(filterSettings,"\n\tLow Expression Filter = ",sep="")
if(is.numeric(minExpFilter)){
  keep <- c(1:nrow(expDataFiltered))[apply(expDataFiltered,1,max)>=minExpFilter]
  removeLowE <- nrow(expDataFiltered)-length(keep)
  expDataFiltered <- expDataFiltered[keep,]
  cat("Done\n",file=diagnosticFile,append=TRUE)
  filterSettings <- paste(filterSettings,minExpFilter," (",removeLowE," genes removed)\n",sep="")
}else{
  cat("Skipped\n",file=diagnosticFile,append=TRUE)
  filterSettings <- paste(filterSettings,"FALSE\n",sep="")
}
```

Running low variance filter

This is the code for the low variance filtering. If minVarFilter is not set to FALSE it will remove the bottom minVarFilter% of genes ranked by variance.

```
cat("Removing low varying genes - ",file=diagnosticFile,append=TRUE)
filterSettings <- paste(filterSettings,"\tLow Variance Filter = ",sep="")
if(is.numeric(minVarFilter)){
  lowVarCut <- quantile(apply(expDataFiltered,1,var),probs=(minVarFilter/100))
  keep <- c(1:nrow(expDataFiltered))[apply(expDataFiltered,1,var)>=lowVarCut]
  removeLowV <- nrow(expDataFiltered)-length(keep)
  expDataFiltered <- expDataFiltered[keep,]
  cat("Done\n",file=diagnosticFile,append=TRUE)
  filterSettings <- paste(filterSettings,minVarFilter,"% (",removeLowV," genes removed)\n",sep="")
}else{
  cat("Skipped\n",file=diagnosticFile,append=TRUE)
  filterSettings <- paste(filterSettings,"FALSE\n",sep="")
}
cat(filterSettings,file=diagnosticFile,append=TRUE)
```

Normalizing the data

This will log₂ transform the expression data if necessary. If alreadyLog2 is set to TRUE the data will not be log₂ transformed.

```
cat("Log2 Transforming expression data - ",file=diagnosticFile,append=TRUE)
if(!alreadyLog2){
  expDataFiltered[expDataFiltered==0] <- 1e-10
```

Code for Fuzzy K-Means clustering

User specified parameters

These seven parameters need to be set before each run of the method.

Parameter	Type	Description
<pre>expDataFiltered <- log2(expDataFiltered/apply(expDataFiltered,1,mean)) expressionData[expressionData==0] <- 1e-10 expressionData <- log2(expressionData/apply(expressionData,1,mean)) cat("Done\n",file=diagnosticFile,append=TRUE) }else{ cat("Skipped\n",file=diagnosticFile,append=TRUE) }</pre>		

Building the distance matrix and clustering

This code will first build a R dist object, `distanceMat`, which holds the distances between each gene and every other gene. The

$$\frac{1 - \rho(i,j)}{2}$$
distance between gene i and j is $\frac{1 - \rho(i,j)}{2}$, where $\rho(i,j)$ is the Pearson correlation between i and j . This distance calculation ensures that genes which are perfectly correlated ($\rho(i,j) = 1$) have a distance of 0, and genes which are perfectly anti-correlated ($\rho(i,j) = -1$), have a distance of 1. See note 4.1.7 about using a different distance metric. The dist object is then used in the `fanny` implementation of the fuzzy K-means algorithm. The result of the clustering (`initClust`), the complete expression data (`expressionData`), and the filtered (and potentially $\log_2$ transformed) expression data (`expDataFiltered`), and other book-keeping variables are then saved into the `resultFile` file.

```
cat("Building distance matrix for clustering - ",file=diagnosticFile,append=TRUE)
distanceMat<-as.dist((1-cor(t(expDataFiltered),method="pearson"))/2)
cat(" Done\n",file=diagnosticFile,append=TRUE)
fuzzyKSettings <- paste("Fuzzy K-Means settings:\n\tK = ",kChoice,"\n\tmemb.exp = ",fuzzyKmemb,sep="")
fuzzyKSettings <- paste(fuzzyKSettings,"\n\tmaxit = ",(nrow(expDataFiltered)*4),"\n",sep="")
cat("Running fuzzy K-means to produce ",kChoice," clusters from ",nrow(expDataFiltered)," genes.",file=diagnosticFile,append=TRUE)
cat(fuzzyKSettings,file=diagnosticFile,append=TRUE)
cat("\n!!! This may take a long time to complete. !!!\n",file=diagnosticFile,append=TRUE)
initClust <- fanny(distanceMat, kChoice, diss = TRUE, memb.exp=
fuzzyKmemb,maxit=nrow(expDataFiltered)*4)
cat("Done clustering!\nSaving results to ",methodResultFile,"\n",file=diagnosticFile,append=TRUE)
save(file=methodResultFile,expressionData,initClust,expDataFiltered,filterSettings,fuzzyKSettings,
dateRun)
cat("Done. Initial clustering completed.\n\n",file=diagnosticFile,append=TRUE)
```

Code for Pattern Identification & Gene Association

User specified parameters

These are the eight parameters which need to be set by the user for the method to identify, collapse and assign genes to patterns.

Parameter	Type	Description
<code>autoSelectClusterCutoff</code>	TRUE/FALSE	This flag tells the method whether or not to determine the <code>clusterCutoff</code> parameter automatically. If set to FALSE the user needs to specify the <code>clusterCutoff</code> parameter.
<code>clusterCutoff</code>	decimal	This is the minimum probability required of a gene belonging to a cluster in order for that gene to be used in building that clusters initial pattern. Ignored if <code>autoSelectClusterCutoff</code> is TRUE.
<code>patternSimilarityCutoff</code>	decimal	Maximum distance between patterns, under which the patterns are collapsed. When using Pearson correlation, 1- <code>patternSimilarityCutoff</code> corresponds to the

Code for Pattern Identification & Gene Association

User specified parameters

These are the eight parameters which need to be set by the user for the method to identify, collapse and assign genes to patterns.

Parameter	Type	Description
pearsonCutoff	decimal	Pearson correlation value above which patterns are collapsed. This sets the Pearson correlation above which a gene is assigned to a final dominant expression pattern.
methodResultFile	string	The filename of the rDump result file from the fuzzy K-means clustering.
userPatternInputFile	string	The filename of the user created patterns to be appended to the set of patterns (note 4.1.11). This file should be in the same format as the input expression data. If FALSE, no user patterns will be appended.
patternOutputFile	string	The filename of the tab-delimited output file, holding the log ₂ transformed dominant expression patterns.
groupOutputFile	string	The filename of the tab-delimited file containing the assignment of genes to each pattern.
diagnosticFile	string	The filename of the file where diagnostic and current status information will be printed.

```

autoSelectClusterCutoff <- TRUE
clusterCutoff <- 0.4
patternSimilarityCutoff <- 0.1
pearsonCutoff <- 0.85
methodResultFile <- "patternIdent_result.rDump"
userPatternInputFile <- FALSE
patternOutputFile <- "patternOutput.txt"
groupOutputFile <- "genesToPatterns.txt"
diagnosticFile <- "patternIdent_diagnostic.txt"

```

Subroutines used in finding and collapsing similar patterns, assigning genes, and writing output

These four subroutines are used by the method to create the initial cluster patterns (`getClustProfiles`), collapse similar patterns (`makeCollapsedProfiles`), calculate the distance from each gene to each pattern (`correlateGenesToProfiles`), and write a table of genes assigned to each pattern (`writeMemberList`).

```

getClustProfiles <- function(expDataIn, clustIn, clusterCutoff){
  numClusters <- ncol(clustIn$membership)
  clustProfiles <- matrix(data=0, ncol=ncol(expDataIn), nrow=numClusters)
  for(i in 1:numClusters){
    genesInClust <- c(1:nrow(expDataIn))[clustIn$membership[,i]>=clusterCutoff]
    if(length(genesInClust)>0){
      if(length(genesInClust)>1){
        clustProfiles[i,]<-as.numeric(apply(expDataIn[genesInClust,], 2, median))
      }else{
        clustProfiles[i,]<-as.numeric(expDataIn[genesInClust,])
      }
    }
  }
  whichCluster <- c(1:numClusters)[apply(clustProfiles, 1, var)>0]
  clustProfiles <- clustProfiles[apply(clustProfiles, 1, var)>0, ]
  return(list(profiles=clustProfiles, whichClust=whichCluster))
}
makeCollapsedProfiles <- function(expDataIn, clustProfsIn,
  clusterCutoff, grouping, origClust, clustIn){
  collapsedProfiles <- matrix(data=0, ncol=ncol(expDataIn), nrow=max(grouping))
  for(i in 1:max(grouping)){

```

Code for Pattern Identification & Gene Association

User specified parameters

These are the eight parameters which need to be set by the user for the method to identify, collapse and assign genes to patterns.

Parameter	Type	Description
<pre> inClust <- c(1:length(grouping))[grouping==i] if(length(inClust)==1){ collapsedProfiles[i,]<-clustProfsIn[inClust,] genesInClust <- c(1:nrow(expDataIn))[clustIn\$membership[,origClust[inClust]]>=0.4] }else{ cat("\tCollapsing profiles ",paste(inClust,collapse="",""), "into new profile ",i,"\n") clustLookAt <- origClust[inClust] genesInClust <- c(1:nrow(expDataIn))[apply(clustIn\$membership[,clustLookAt],1,max)>= clusterCutoff] collapsedProfiles[i,]<-apply(expDataIn[genesInClust,],2,median) } } return(collapsedProfiles) } correlateGenesToProfiles <- function(expDataIn,profilesIn){ profileCor <- matrix(data=0,ncol=nrow(profilesIn),nrow=nrow(expDataIn)) for(i in 1:nrow(profilesIn)){ profileCor[i,] <- apply(expDataIn,1,cor,y=profilesIn[i,]) cat("\tCorrelating genes to profile ",i," out of ",nrow(profilesIn),"\n") } rownames(profileCor)<-rownames(expDataIn) colnames(profileCor)<-rownames(profilesIn) return(profileCor) } writeMemberList <- function(patternCor,minDist,fileOut){ maxSize <- max(apply(patternCor>=minDist,2,sum)) tempMat <- matrix(data="",ncol=ncol(patternCor),nrow=maxSize) colnames(tempMat)<-colnames(patternCor) for(i in 1:ncol(tempMat)){ geneList <- rownames(patternCor)[patternCor[,i]>=minDist] if(length(geneList)>0){ tempMat[1:length(geneList),i]<-geneList } } write.table(file=fileOut,tempMat,sep="\t",quote=FALSE,row.names=FALSE,col.names=colnames(tempMat)) } </pre>		

Creation of initial patterns

This code loads the results from the fuzzy K-means clustering and will build an initial set of patterns. If `autoSelectClusterCutoff` is TRUE, then the method will set `clusterCutoff` to a value at which the average gene is assigned to one cluster. The method then finds genes which do not belong to any cluster above a probability of `clusterCutoff` and flags those genes. The `getClustProfiles` subroutine then builds the initial set of patterns by finding the median (by column) expression pattern of all genes assigned to each cluster above `clusterCutoff`. The patterns are then stored in the `clustProfiles` variable.

```

cat("Starting Pattern Identification & Gene Assignment\n",file=diagnosticFile)
load(clusterResultFile)
if(autoSelectClusterCutoff){
  clusterCutoff <- initClust$membership[sort(initClust$membership,index.return=TRUE,decreasing=TRUE)
  $ix[nrow(initClust$membership)]]
}
cantAssignGenes<-(apply(initClust $membership,1,max)<clusterCutoff)
goodGenes<-(apply(initClust $membership,1,max)>=clusterCutoff)
initClust$cluster[cantAssignGenes]<-0
initClust$membership[cantAssignGenes,]<-0
cat("Number of genes assigned to a cluster above ",clusterCutoff,"
=",file=diagnosticFile,append=TRUE)
cat(sum(goodGenes),"/",length(initClust$cluster),"\n",file=diagnosticFile,append=TRUE)
cat("Number of genes not assigned to a cluster above ",clusterCutoff,"
=",file=diagnosticFile,append=TRUE)
cat(sum(cantAssignGenes),"/",length(initClust$cluster),"\n",file=diagnosticFile,append=TRUE)
cat("Generating initial set of patterns.\n",file=diagnosticFile,append=TRUE)
clustProfiles <- getClustProfiles(expDataFiltered,initClust,clusterCutoff)

```

Code for Pattern Identification & Gene Association

User specified parameters

These are the eight parameters which need to be set by the user for the method to identify, collapse and assign genes to patterns.

Parameter	Type	Description
<code>whichClusters</code>	<code><- clustProfiles\$whichClust</code>	
<code>clustProfiles</code>	<code><- clustProfiles\$profiles</code>	

Collapsing similar patterns

This code takes the initial set of patterns and collapses those that are similar to each other. The method calculates the distance (1-Pearson correlation) between all the patterns and then performs single-linkage hierarchical clustering using that information. The resulting tree is cut at a height of `patternSimilarityCutoff`, and patterns which are in the same cluster are collapsed in the `makeCollapsedProfiles` subroutine. The final set of unique dominant expression patterns is then written to `patternOutputFile` in the same format as the input expression data (see note 4.1.13 for output formats).

```
cat("Calculating distances between each pattern.\n",file=diagnosticFile,append=TRUE)
profileCor <- cor(t(clustProfiles))
diag(profileCor)<-NA
hClust <- hclust(as.dist(1-profileCor),method="single")
collapseGrp <- cutree(hClust,h=patternSimilarityCutoff)
cat("Collapsing similar patterns.\n",file=diagnosticFile,append=TRUE)
finalProfiles<-makeCollapsedProfiles(expDataFiltered,clustProfiles,
clusterCutoff ,collapseGrp,whichClusters,initClust)
rownames(finalProfiles)<- paste("Pattern_",1:nrow(finalProfiles),sep="")
colnames(finalProfiles)<-colnames(expDataFiltered)
colnames(finalProfiles)[1] <- paste("\t",colnames(finalProfiles)[1],sep="")
if(is.character(userPatternInputFile)){
cat("Adding user defined patterns from \"",userPatternInputFile,"\".
\n",file=diagnosticFile,append=TRUE)
userPatterns <- read.delim(file=userPatternInputFile,sep="\t",header=TRUE)
tempFinal <- matrix(data=0,ncol=ncol(finalProfiles),nrow=nrow(finalProfiles)+nrow(userPatterns))
tempFinal[1:nrow(finalProfiles),]<-finalProfiles
tempFinal[(nrow(finalProfiles)+1):nrow(tempFinal),]<-as.matrix(userPatterns[,
2:ncol(userPatterns)])
rownames(tempFinal) <- c(rownames(finalProfiles),userPatterns[,1])
colnames(tempFinal) <- colnames(finalProfiles)
finalProfiles <- tempFinal
}
cat("Writing patterns to output file.\n",file=diagnosticFile,append=TRUE)
write.table(file=patternOutputFile,finalProfiles,quote=FALSE,sep="\t")
```

Assignment of genes to patterns

This code takes the final set of unique dominant expression patterns and calculates the pearson correlation between each gene used in the clustering (see note 4.1.12 for assigning genes not used in the clustering) to each pattern. This information is passed to the `writeMemberList` subroutine, and it creates a file (`groupOutputFile`), which is table where each column is a pattern, and each row contains the name of a gene which pearson correlates to that patterns above `pearsonCutoff`. Some genes may appear in multiple columns, and some may not appear at all. Finally the method deletes some bookkeeping and temporary variables and stores the rest of the results of the analysis in the `rDump` file specified by `methodResultFile`.

```
cat("Assigning genes to patterns.\n",file=diagnosticFile,append=TRUE)
geneToPatternCor <- correlateGenesToProfiles(expDataFiltered,finalProfiles)
cat("Writing gene to pattern output file.\n",file=diagnosticFile,append=TRUE)
writeMemberList(geneToPatternCor,pearsonCutoff,groupOutputFile)
rm(cantAssignGenes,goodGenes,profileCor,hClust,collapseGrp)
save(file=methodResultFile,list=ls())
cat("Pattern Identification and Gene Assignment completed.\n",file=diagnosticFile,append=TRUE)
```