FISA: Fast Iterative Signature Algorithm for the analysis of large-scale gene expression data

Seema Aggarwal
Department of Computer Science
University of Delhi
saggarwal@mh.du.ac.in
and
Neelima Gupta
Department of Computer Science
University of Delhi
ngupta@cs.du.ac.in

May 25, 2007

Abstract

One approach to reduce the complexity of the task in the analysis of large scale genome-wide expression is to group the genes showing similar expression patterns into what are called *transcription modules* (TM). A TM is defined as a set of genes and a set of conditions under which these genes are most tightly co-expressed. Most of the existing algorithms compute non-overlapping TMs whereas a gene may be responsible for more than one cellular activity and hence must be included in more than one TMs. Existing algorithms to compute overlapping *transcription modules* either require prior biological information of co-regulated genes or start with a totally random input gene seed. In this paper, we present an algorithm in which we generate an intelligent gene seed from the expression data itself, find out the conditions which are most favourable to these genes and then iteratively improve the TM. This eliminates the need to have a prior information about co-regulated genes and since the input genes are chosen intelligently the algorithm converges fast. In fact, we generate several such seeds and iterate on all of them simultaneously. Experimental results were obtained for synthetic data as well as for the expression data from the yeast Saccharomyces cerevisiae. TMs obtained for the yeast data were confirmed to contain $30\% - 50\%$ of the genes showing similar cellular functionality using Gene Ontology database.

1 Introduction

Regulatory and metabolic functions of cells are carried out by several biochemical components interacting with each other. Many such cellular processes are regulated by enzymes translated on binding of one or more transcription factors to short DNA sequences called *motifs* in the upstream regions of the process genes. One way to find out the genes responsible for a cell function is to study the DNA sequences of the genes to extract a common pattern among them. This problem is called 'motif finding'. Another approach is to study their expression patterns. When a group of genes binds to a set of *transcription factors* they show similar expression patterns. With the help of microarray experiments biologists are able to study the expression of thousands of genes under a large number of conditions simultaneously. The large scale of the data makes it challenging to analyse it to extract any biologically significant information from it.

One way to reduce the complexity of the task is to classify the data into clusters of genes which show similar expression levels. Once the genes have been clustered, their sequences can be studied for the presence of one or more motifs and hence determine the *transcription factors* that bind to them.

Standard Clustering algorithms like k-means clustering work well for small data sets but fair poorly when the number of experimental conditions is large as they cluster the genes based on their expressions under all the conditions whereas the cellular processes are generally affected by a small subset of conditions. Most of the other conditions which do not contribute to the cellular process add to the background noise. Moreover, these algorithms compute non-overlapping clusters i.e. a gene belongs to at most one cluster whereas in fact a gene may be responsible for several cellular activities and hence must be included in more than one clusters. The first problem is addressed in *projected clustering*. In *projected clustering* data points (genes in our case) are projected onto a relevant set of dimensions (conditions in our case) and a cluster is defined as a set of data points and a set of dimensions that are most relevant to these data points. However, algorithms for *projected clustering* [YCN04, AGGR98, APW$^+$99, AY00, YM05, PJAM02] also compute non-overlapping clusters. The clusters overlap on the conditions but not on the genes. Hence very few of them have been used to cluster the gene expression data [YCN04].

In [CC00] Cheng and Church introduced the notion of biclustering in which the clusters are defined to be a set of genes and a set of conditions under which these genes are most tightly regulated. Though their algorithm discover non-overlapping bi-clusters they suggest that with small perturbation in the expression values of some selected genes they would be able to find overlapping bi-clusters. Their claim has been contradicted by Wang etal in [WWYY84]. In [GLD00] Getz etal gave an algorithm which uses one-way clustering once to cluster the genes over all the conditions and in each of the clusters so obtained, it recursively applies one-way clustering again to cluster the conditions over the genes. It repeats this process creating a tree of modules. In [KBCG03], Chang etal gave an algorithm that simultaneously clusters genes and conditions only for an expression matrix having a checker-board pattern. Their method is based on Singular Value Decomposition technique from linear algebra on the expression matrix. The checker-board structure is reflected in the singular values of the expression matrix.

In [IFB$^+$02] Ihmels etal have given the name *transcription modules* (TM) to *bi-clusters*. They call the set of genes and the set of conditions obtained for a module as a *signature* of the module and hence their algorithm as *Signature Algorithm*. A TM is a set of genes and conditions where the genes in the TM are most similarly expressed under the conditions of the TM and the conditions of the TM generate the most similar expression pattern for the genes of the TM. Starting from an input set of genes, they compute the set of experimental conditions under which these genes are co-regulated most tightly. They compute the condition scores as the average change in the expression of the input genes for each condition. Noisy conditions are filtered out by keeping only the conditions with a large (absolute) score and discarding the ones with low scores. It then computes the gene score, for each gene, as the weighted average change in the expression over these conditions, using condition scores as the weights and selects the genes with high gene scores. *Signature Algorithm* works well if the input genes are co-regulated. In [BIB03] Bergmann etal have given an algorithm that eliminates the requirement of the prior biological information of co-regulated genes. Starting with a random input seed their algorithm iteratively improves the set of conditions and the set of genes. Their algorithm is called *Iterative Signature Algorithm* (ISA). Kloster etal [KTW05] have extended their work to find orthogonal modules. Once a module is found, they delete its component from the expression matrix and compute the next module in a direction orthogonal to the first one.

In this paper, we present a better way to overcome the problem of specifying co-regulated gene seed. Instead of starting with a completely random input genes, we try to generate a seed consisting of genes which are co-regulated or similar in some sense from the expression data itself. We then obtain the set of conditions which are most significant for these genes. We then improve the quality of the TM iteratively like ISA. Since the input genes are not totally random, our algorithm converges much faster than ISA. Moreover, we compute several TMs simultaneously.

The main idea of our algorithm is to create a set of k medoid genes randomly. Medoids are chosen far apart from each other so that if the TMs are scattered, we tend to pick one medoid from several TMs (assuming k is large). For each medoid m we compute a set of genes which are similar to m in some sense. We then take this set as the seed to compute the set of conditions which are most significant for these genes and iterate. The iterative algorithm is run on all the seeds giving us k TMs simultaneously. A potential problem in such an approach is the choice of the value of k. In most of the previous work where this approach has been used [APW$^+$99] the nature of clusters change as k increases or decreases with the size of the clusters being large when k is small and real clusters splitting in too small clusters for large k. For example in [APW$^+$99] the radius of each medoid (defined as the minimum distance of a medoid from all other medoids) is calculated and a point (gene) is assigned to a cluster if its distance from the medoid of the cluster is the least. If k is large the medoids will be close to each other and the radii of the medoids will be small, thus a real cluster is split if two medoids are chosen from it. A small value of k results in small number of large clusters missing out some clusters. To eliminate this problem, we introduce a similarity threshold s_g, which defines the extent upto which the genes are considered to be similar to a given medoid. By doing so, the impact of varying k on the initial gene seeds is eliminated. Hence, the problem of splitting is eliminated for an over-estimated value of k. For large k, when two or more medoids are chosen from a real TM, instead of splitting the TM, all the medoids lead to the same TM. We say that two TMs are same if they overlap in at least 80% of the genes.

We ran our algorithm on a synthetic data as well as Yeast data. Synthetic data was created for two overlapping modules. For $k \geq 2$ we were able to obtain both the modules and the module corresponding to their intersection. We verified the modules obtained for the Yeast data by checking their functionality from Gene Ontology database (http://www.yeastgenome.org/). Most of the TMs obtained contained 30 to 50 percent of the genes showing common functionality.

2 Definitions and Notations

We assume that the expression data from microarray experiments is given by gene expression matrix E . The $(i, j)^{th}$ entry of the matrix denoted by E_{ij} is the log-fold expression change of gene g_i under the j^{th} experiment. The matrix E is thus an $n\ x\ m$ matrix, where n is the total number of genes and m is the total number of conditions. A *transcription module* M is defined by a set of genes and a set of conditions such that these genes are more tightly co-expressed under these conditions as compared to other conditions and these conditions are more favourable to these genes as compared to other genes. We denote M by the set $\{G, C\}$, where $G = (g_1, g_2, \ldots g_n)$ and $C = (c_, c_2, \ldots c_m)$ are vectors of genes and conditions respectively such that $g_i, i = 1, n$ and $c_j, j = 1, m$ are non-zero if and only if g_i and c_j are in M. The problem is to find M in E.

Denote the distance between a pair of genes g_i and g_j by $\delta(g_i, g_j)$, where $\delta(g_i, g_j) = \sum_{k=1,m} |E_{ik} - E_{jk}|$. Let S be a set of genes. Denote the distance of a gene g from S by $\delta(g, S)$, then $\delta(g, S) = min_{g' \in S}\delta(g, g')$.

To compute the initial set of genes, we select k medoids randomly. Let $T = \{m_1, m_2 \ldots m_k\}$ represent the set of medoids. Let L_i be the set of genes which are similar to medoid m_i. The notion of similarity is defined later.

Due to experimental errors, some genes may show higher and some may show lower expression levels as compared to their real expression values. Similarly, a gene may show higher expression values under some conditions and lower expression values under some other conditions as compared to their real expression values. To handle this situation the expression matrix is normalized once over all the genes and once over all the conditions. Let E_G be the matrix obtained by normalizing the expression matrix E over all genes. Similarly E_C is the matrix obtained by normalizing the expression matrix E over all conditions.

To find out the relevant conditions for a given set of genes, we find out how each condition scores for these genes and keep only the ones with high scores. Similarly, to find out the relevant genes for a given set of conditions, we find out how each gene scores for these conditions and keep only the ones with high scores. Let S^c denote the vector of condition scores and S^g the vector of gene scores.

3 Overview of Iterative Signature Algorithm

The *Iterative Signature Algorithm* normalizes the expression matrix E to obtain the two matrices E_G and E_C to eliminate any experimental errors such that $\sum_i E_G^{ij} = 0$, $\sum_i (E_G^{ij})^2 = 1$ for each condition c_j and $\sum_j E_C^{ij} = 0$, $\sum_j (E_C^{ij})^2 = 1$ for each gene g_i. Starting with a random input seed $\hat{g}$, it computes, for each condition, the average condition score over the input genes i.e. it computes $S^c = E_G \cdot \hat{g}$. The i^{th} component of S^c is set to zero $\frac{S_i^c - \mu(S^c)}{\sigma(S^c)} < t_c$ where t_c is the condition threshold and, $\mu(x)$ and $\sigma(x)$ denote the mean and variance of x. It then computes, for each gene, the average gene score over the non-zero components of S^c i.e. it computes $S^g = E_C \cdot S^c$. The i^{th} component of S^g is set to zero if $\frac{S_i^g - \mu(S^g)}{\sigma(S^g)} < t_g$ where t_g is the gene threshold. The algorithm then iterates until either (i) number of iterations exceeds a certain number or (ii) $\frac{|g^{n+1} - g^n|}{|g^{n+1} + g^n|} < \epsilon$ where g^n is the gene vector obtained in the n^{th} iteration.

4 FISA : Fast Iterative Signature Algorithm

In this section, we present an algorithm to compute *transcription modules* in gene expression data given as an $n\ x\ m$ expression matrix E, where n is the total number of genes and m is the total number of conditions. The main idea of our algorithm is to create a set of k medoids randomly. Medoids are chosen far apart from each other. For each medoid m we compute a set of genes which are similar to m in some sense. These are then taken as the input gene seeds to compute the k TMs simultaneously. The way the medoids are chosen, irrespective of the first medoid, for an appropriate value (an over-estimated value) of k, the medoids from other TMs are also chosen. The algorithm works in three phases.

Phase 1: Find a set T of k medoid genes which are far apart from each other. The first medoid is chosen randomly. The second medoid is chosen to be the gene which is farthest from the first medoid. The third medoid gene is chosen which is farthest from the previous medoids and so on. In this way we pick up medoid genes which are well separated from each other. Also, if the TMs are scattered in the expression matrix, we tend to pick one medoid gene from every TM (assuming k is large enough). The procedure Compute-medoid-set-T() computes the set T of medoids.

```
Compute-medoid-set-$T$()
  1. $T = m_1\{m_1$ is the first medoid gene which is generated randomly$\}$.
  2. For $indx = 2$ to $k$
     (a) For each gene $g \notin T$
             Compute distance $\delta(g, T)$.
     (b) Let $m = argmax_g\{\delta(g, T)\}$.
     (c) Include $m$ in $T$.
```

Phase 2: For every medoid gene m_i found in Phase 1, compute the subset of genes L_i which are similar to m_i. For computing L_i, we compute the distance of each gene from m_i over all the conditions. Let X_{ij} denote the distance between gene g_j and m_i. Let μ_i and σ_i^2 denote the mean and variance of these distances i.e. $\mu_i = \Sigma_j X_{ij}/n$ and $\sigma_i^2 = \Sigma_j (X_{ij} - \mu_i)^2/(n-1)$ then $L_i = \{g_j : (\frac{X_{ij}-\mu_i)}{\sigma_i} < s_g\}$. If $|L_i| < .1\%$ of total number of genes then discard m_i and L_i. Remaining L_i's serve as an input seed for Phase 3. Given the set T of medoid genes, the procedure Compute-gene-seeds-L_i(T) computes k gene seeds $L_i, i = 1, k$.

```
Compute-gene-seeds-$L_i$($T$)
For every $m_i \in T$ do
  1. for each gene $g_j$ in $E$ do
         compute $X_{ij} = \delta(g_j, m_i)$
  2. compute $\mu_i = \Sigma_j X_{ij}/n, \sigma_i = \sqrt{\Sigma(X_{ij} - \mu_i)^2/(n-1)}$
  3. for each gene $g_j$ in $E$ do
     (a) compute $x_j = (X_{ij} - \mu_i)/\sigma_i$.
     (b) if $x_j < s_g$ put $g_j$ in $L_i$.
```

Phase 3: In this phase, for each L_i computed in Phase 2, we compute, for each condition, the average condition score over the genes in L_i i.e. we compute $S^c = E_G \cdot L_i$. Let $\hat{S}^c$ be the condition vector whose i^{th} component is non-zero only if $\frac{S_i^c - \mu(S^c)}{\sigma(S^c)} > t_c$ where t_c is the condition threshold and, $\mu(x)$ and $\sigma(x)$ denote the mean and variance of x. We then compute, for each gene, the average gene score over the non-zero components of $\hat{S}^c$ i.e. we compute $S^g = E_C \cdot \hat{S}^c$. Let $\hat{S}^g$ be the gene vector whose i^{th} component is non-zero only if $\frac{S_i^g - \mu(S^g)}{\sigma(S^g)} > t_g$ where t_g is the gene threshold. The algorithm then iterates until either (i) number of iterations exceeds a certain number or (ii) $\frac{|g^{n+1} - g^n|}{|g^{n+1} + g^n|} < \epsilon$ where g^n is the gene vector obtained in the n^{th} iteration. The procedure Compute-TMs() computes the TMs corresponding to gene seeds L_i.

```
Compute-TMs()
Let $M_i = \{G_i, C_i\}$ denote the TM obtained for seed $L_i$.

  1. Normalize genes over conditions to obtain the matrix $E_C$.
     $E_C^{ij} = \frac{E^{ij} - \frac{\sum_j E^{ij}}{n}}{|E^{ij} - \frac{\sum_j E^{ij}}{n}|}$ for each gene $g_i$.
  2. Normalize conditions over genes to obtain the matrix $E_G$.
     $E_G^{ij} = \frac{E^{ij} - \frac{\sum_i E^{ij}}{n}}{|E^{ij} - \frac{\sum_i E^{ij}}{n}|}$ for each condition $c_j$.
  3. For each $L_i$ do
     $\hat{S}^g = L_i$
     Repeat until the terminating condition is met
       (a) Compute $S^c = E_G \cdot \hat{S}^g$. $\hat{S}^c = S^c$.
       (b) Apply condition threshold on $\hat{S}^c$.
       (c) Compute $S^g = E_C \cdot \hat{S}^c$. $\hat{S}^g = S^g$.
       (d) Apply gene threshold on $\hat{S}^g$.
```

We have used two gene thresholds because their effects are different. s_g, the similarity threshold is used to compute the genes which are similar to a medoid whereas t_g is used to select genes with high gene scores. Thus as we decrease s_g we get less genes similar to the medoid whereas when we decrease t_g we get more genes of high scores.

Pearson's coefficient was used to determine the reliability of the TMs. The procedure FISA() summarizes our algorithm.

```
FISA()
  1. Compute-medoid-set-$T$()
  2. Compute-gene-seeds-$L_i(T)$. Discard $m_i$ and $L_i$ if
     $|L_i| < 0.1\%n$.
  3. Compute-TMs()
```

Effect of increasing k: As k increases, TMs start replicating. For large k, more than one medoids may be chosen from a real TM. However, it does not result in splitting as L_i's and S^g's depend on the threshold parameters s_g and t_g respectively rather than the density of the medoid set.

Effect of varying s_g: As we increase s_g, L_i's have large overlap and hence they start converging to TMs having large overlaps. As a result, the number of unique TMs reduce. As we decrease s_g, L_i's become small. We discard a medoid and its corresponding seed if L_i becomes too small.

Effect of varying t_g: The changing values of t_g affect the granularity of the TMs.

5 Experimental Results on synthetic data

We created the synthetic expression data for two overlapping modules. Each module consisted of 55 genes and 30 conditions as shown in figure 1.

For $k = 10, s_g = 0.5, t_g = 0.5$ and $t_c = 0.5$ we obtained three TMs, a TM consisting of genes $1 - 45$ and conditions $1 - 20$, a TM consisting of genes $56 - 100$ and conditions $31 - 50$ and, a TM consisting of the intersection of $M1$ and $M2$ i.e. genes $46 - 55$ and conditions $21 - 30$. As we decrease s_g we continue to get the same TMs. In order to get TMs $M1$ and $M2$ we decreased both t_g and t_c in steps of 0.1. At $t_g = 0.1$ and $t_c = 0.2$, we get complete TMs, one consisting of genes $1 - 55$ and conditions $1 - 30$ and the other consisting of genes $46 - 100$ and conditions $21 - 50$. By reducing t_c further to - 0.1, we get $M1 \cup M2$. The value of s_g is kept fixed at 0.5 in all these experiments. As we increase k keeping s_g, t_g and t_c fixed to 0.5, 0.1 and 0.2 respectively no new TMs are obtained. As we increase s_g from 0.5 to 1 in the steps of 0.1, with $t_g = 0.1$ and $t_c = 0.2$, we continue to get both the modules $M1$ and $M2$ and finally for $s_g = 1$ we get only one TM which is the intersection of $M1$ and $M2$.

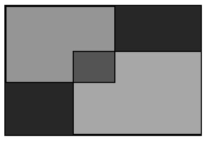

Figure 1: Expression matrix for synthetic data showing two overlapping modules.

6 Experimental Results on Yeast data

Gene expression data for Saccharomyces cerevisiae was downloaded from the site http://www.weizmann.ac.il. The data contains expression profiles of 6206 genes under 1011 conditions.

We ran our algorithm for varying k on Yeast data. We observed that as k increases, TMs start repeating. For large k, most of the TMs repeat many number of times and very few new TMs are found. The following table summarizes the results for varying s_g. We observe that the number of unique TMs reduce as we increase s_g. The table also summarizes the cellular activity with which the maximum number of genes within each module is associated. The functional significance (ca: catalytic activity, mfu: molecular function unknown, sma: structural molecular activity) of the modules was obtained using Gene Ontology database.

tg=2 tc=2 k=150											
sg=1			**sg=2**			**sg=3**			**sg=6**		
sno.	max(%)	func.	sno.	max(%)	func.	sno.	max(%)	func.	sno.	max(%)	func.
1.	45.1	mfu	1.	40.7	mfu	1.	40.7	mfu	1.	28	mfu
2.	47.7	ca	2.	43.6	ca	2.	43.6	ca	2.	46	ca
3.	46.8	mfu	3.	28.8	mfu	3.	42.7	ca	3.	44.6	sma
4.	39.5	mfu	4.	45.1	ca	4.	45.1	ca	4.	44.8	sma
5.	53.5	mfu	5.	43.8	sma	5.	43.8	sma	5.	43.8	sma
6.	40.7	mfu	6.	42.8	ca	6.	38.6	mfu	6.	31.9	mfu
7.	43.6	ca	7.	43.5	ca	7.	42.8	ca	7.	42.6	sma
8.	28.8	mfu	8.	37.3	mfu	8.	45.6	sma			
9.	45.1	mfu	9.	48	mfu						
10.	37.3	mfu	10.	38.3	mfu						
11.	43.8	sma									
12.	42.8	ca									

7 Conclusion and Future Work

One of the problems faced in computing overlapping *transcription modules* for the gene expression data is to specify an input gene seed. The existing algorithms either require a prior knowledge of some co-regulated genes or work with a totally random input gene seeds. In this paper, we have presented a method to compute initial gene seed containing genes which are similar in some sense. Our algorithm does not require any prior knowledge of co-regulated genes but rather extract this information from the expression data itself and then computes the set of conditions and genes iteratively. Since the input seed is chosen intelligently, the iterative algorithm converges fast. We compute several seeds and compute modules for all of them simultaneously. We obtained experimental results for synthetic data created to contain two overlapping modules and we were able to obtain all the TMs by varying the parameters. We also showed that the TMs do not split for large k. TMs were also obtained experimentally for the yeast data and they were confirmed to contain $30\% - 40\%$ of the genes showing similar cellular functionality using Gene Ontology database.

8 Acknowledgements

We thank Jan Ihmels and Naama Barkai, Weizmann Institute of Science, Israel for allowing us to download the Yeast expression data from their website.

References

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic Subspace Clustering Of High Dimensional Data For Data Mining Application. In *ACM SIGMOD Int'l Conf. Management of Data*, 1998.

[APW$^+$99] Charu C. Aggarwal, Cecilia Procopiuc, Joel L. Wolf, Philip S. Yu, and Jong Soo Park. Fast Algorithms For Projected Clustering. In *ACM SIGMOD Int'l Conf. Management of Data*, 1999.

[AY00] Charu C. Aggarwal and Philip S. Yu. Finding Generalised Projected Clusters In High Dimensional Spaces. In *ACM SIGMOD Int'l Conf. Management of Data*, 2000.

[BIB03] S. Bergmann, J. Ihmels, and N. Berkai. Iterative signature algorithm for the analysis of large-scale gene expression data. In *Physical Review*, volume 67, pages 1–18. American Physical Society, 2003.

[CC00] Y. Cheng and G. M. Church. Byclustering Of Gene Expression Data. In *System Molecular Biology*, volume 8, pages 1–93. System Molecular Biology, 2000.

[GLD00] Gad Getz, Erel Levine, and Eytan Domany. Coupled Two-Way Clustering Analysis Of Gene Microarray Data. In *Cell Biology*, volume 97, pages 12079–12084. PNAS, 2000.

[IFB$^+$02] J. Ihmels, G. Friedlander, S. Bergmann, Y. Ziv O. Sarig, and N. Barkai. Revealing Modular Organization In The Yeast Transcription Network. In *Natural Genetics*, volume 31, pages 1–370. Nature Publishing Group, 2002.

[KBCG03] Yuval Kluger, Rones Basri, Joseph T. Chang, and Mark Gerstein. Spectral Biclustering Of Microarray Data: Coclustering Genes And Conditions. In *Genome Research*. Cold Spring Harbor Laboratory Press, 2003.

[KTW05] M. Kloster, C. Tang, and N.S. Wingreen. Finding regulatory modules through large-scale gene-expression data analysis. In *Bioinformatics*, volume 21, pages 1172–1179, 2005.

[PJAM02] Cecilia M. Procopiuc, Micheal Jones, Pankaj K. Agarwal, and T. M. Murali. Monte Carlo Algorithm For Fast Projective Clustering. In *ACM SIGMOD Int'l Conf. Management of Data*, 2002.

[WWYY84] H. Wang, W. Wang, J. Yang, and P. S. Yu. Clustering by Pattern Similarity in Large Data Sets. In *Bull. Math. Biol. 46*, pages 515–527. ACM Press, 1984.

[YCN04] Kevin Y. Yip, David W. Cheung, and Micheal K. Ng. HARP: A Practical Projected Clustering Algorithm. In *IEEE Transactions On Knowledge And Data Engineering*, volume 16. IEEE Computer Society, 2004.

[YM05] Man Lung Yiu and Nikos Mamoulis. Iterative Projected Clustering By Subspace Mining. In *IEEE Transactions On Knowledge And Data Engineering*, volume 17. IEEE Computer Society, 2005.