Genes@Work: an efficient algorithm for pattern discovery and multivariate feature selection in gene expression data

Jorge Lepre, J. Jeremy Rice, Yuhai Tu and Gustavo Stolovitzky*

IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA

Received on July 21, 2003; revised on October 30, 2003; accepted on October 31, 2003
Advance Access publication February 5, 2004

ABSTRACT
Motivation: Despite the growing literature devoted to finding differentially expressed genes in assays probing different tissues types, little attention has been paid to the combinatorial nature of feature selection inherent to large, high-dimensional gene expression datasets. New flexible data analysis approaches capable of searching relevant subgroups of genes and experiments are needed to understand multivariate associations of gene expression patterns with observed phenotypes.

Results: We present in detail a deterministic algorithm to discover patterns of multivariate gene associations in gene expression data. The patterns discovered are differential with respect to a control dataset. The algorithm is exhaustive and efficient, reporting all existent patterns that fit a given input parameter set while avoiding enumeration of the entire pattern space. The value of the pattern discovery approach is demonstrated by finding a set of genes that differentiate between two types of lymphoma. Moreover, these genes are found to behave consistently in an independent dataset produced in a different laboratory using different arrays, thus validating the genes selected using our algorithm. We show that the genes deemed significant in terms of their multivariate statistics will be missed using other methods.

Availability: Our set of pattern discovery algorithms including a user interface is distributed as a package called Genes@Work. This package is freely available to non-commercial users and can be downloaded from our website (http://www.research.ibm.com/FunGen).

Contact: gustavo@us.ibm.com

INTRODUCTION
Recent advances in DNA microarray technology have opened an important window into the internal workings of the cell. Microarrays allow the measurement of the expression level of thousands of genes simultaneously with a single experiment (Brown and Botstein, 1999; Lockhart et al., 1996). While the expression level of all genes in the cell is not a complete description of the state of the cell, expression data have been shown to contain information about the molecular mechanism underlying different cell types and behavior. For example, the expression levels of a gene under varied conditions contain information about the functional category of the gene (Brown et al., 2000; Eisen et al., 1998; Mateos et al., 2002).

The measured gene expression profile across all the genes also contains information about the type of tissue or ‘phenotype’ of the cell. For example, a widely studied phenotype is that of cancerous cells. Many successful works have been published demonstrating how cancer phenotypes can be characterized by gene expression profiles (Alon et al., 1999; Furey et al., 2000; Golub et al., 1999; Guyon et al., 2002; Ross et al., 2000 and many others).

Many methods have been proposed for selecting genes that behave differentially in two tissue types (for a review, see Stolovitzky, 2003). To discover differentially expressing genes between two phenotypes, we propose a combinatorial pattern discovery algorithm that we call Genes@Work. We have previously described similar constructs in the context of phenotype classification (Califano et al., 2000). Our pattern discovery method entails searching for patterns that differentiate one particular phenotype from another phenotype chosen as a reference or control. Each pattern is a subgroup of genes observed to act consistently over a subset of the experiments in the phenotype. Straightforward methods of finding such patterns become computationally intractable in large datasets by requiring enumeration of an exponentially growing space of sub-patterns (e.g. finding all subsets of genes and all subsets of experiments that satisfy some given pattern parameters). In this study, we present a computationally efficient, deterministic and exhaustive algorithm that avoids searching the complete combinatorial space. Other efficient algorithms have been described for the discovery of patterns in biological sequences (Califano, 2000; Parida et al., 2000; Rigoutsos and Floratos, 1998), but these have not been optimized for the task of pattern discovery in the context of gene expression research.

Other pattern-based approaches have been reported recently in the literature. In Li et al. (2003) and Li and Wong (2002), the
data is discretized before patterns are found. In our approach, we will explicitly handle the continuous nature of the gene expression data. Our definition of pattern can be interpreted as a cluster of samples over a subset of genes, an idea that has been referred to in the recent literature as biclustering. Recent biclustering work includes Cheng and Church (2000) (who used greedy algorithms to identify groups of genes that fluctuate in unison in a subset of samples), Getz et al. (2000) (whose biclusters consist of pairs of stable gene and sample clusters generated by iterations of hierarchical clustering), Tanay et al. (2002) (who used graph theory and statistics to find subsets of genes that exhibit a similar expression pattern across a subset of conditions), Lazzeroni and Owen (2002) (who define ‘plaid’ models that describe the gene expression matrix as a linear function of their bicluster parameters), and Bergmann et al. (2003) (who iteratively refine sets of genes and conditions until they match the definition of a special kind of bicluster that they call transcription modules). There are important differences between these methods and our approach. All these clustering methods employ heuristics and are not guaranteed to find all patterns. In contrast, we will present an exhaustive pattern discovery algorithm. All the methods mentioned above are based on unsupervised clustering, whereas our approach is supervised. The similarity in gene expression in our method is measured using a novel metric. Our patterns are maximal in genes and samples. Finally, an important feature of our approach (also important in Tanay et al. (2002)) is that for each pattern we compute a statistical significance, an aspect that facilitates the segregation of interesting from spurious patterns.

The paper is organized as follows. First, we present our approach to the normalization of the phenotype samples in terms of the control samples, the definition of a pattern and the statistical significance of patterns. Then, we present in detail the algorithm used to identify the gene expression patterns. The performance and scalability of the algorithm is subsequently explored. Next, the utility of the algorithm is demonstrated with a specific example to discover genes in two independently collected sets of lymphoma data. We finish the paper with some concluding remarks.

**SYSTEMS AND METHODS**

**Normalization of gene expression**

In the following discussion, we assume that we are comparing gene expression samples from two distinct phenotypes. Samples of one such phenotype will be referred to as the ‘control’ samples, and the others simply as the ‘phenotype’ samples. Finding consistent gene expression differences between these two sample sets suggest a functional role in the conditions under consideration. However, finding consistent differences requires enough number of samples to find signal above the variability with sufficient statistical significance. For example, gene expression measurements contain many sources of variability including sample purity, sample preparation variability and the measurement process (Tu et al., 2002; Workman et al., 2002).

In view of its variability, it is natural to conceptualize the gene expression of a gene as a continuous random variable whose probability density function describes the distribution of gene expression levels across different samples. One might try modeling all possible gene dependencies by a joint distribution, but this requires larger datasets than those generally available. Therefore, our method considers probability density functions of individual genes. For example, consider the expression level of samples of a gene $g$ in the control set and its probability distribution estimate $F_g(x)$ (Fig. 1a). Using this estimate, we define the normalized gene expression for gene $g$ as:

$$\mu(x) \equiv \hat{F}_g(x).$$  \hspace{1cm} (1)

A useful property of this normalization is that, if the gene expression $x$ is distributed as in the control set, then the normalized values will be uniformly distributed in the interval $[0, 1]$ (Fig. 1a, for a representation of this property). This property allows us to detect changes in gene expression distribution by detecting how the normalized expressions deviate from a uniform distribution. In Figure 1b, a hypothetical distribution for the gene expression levels on the phenotype samples is compared with the distribution on the control samples. The phenotype distribution happens to be bimodal in Figure 1b, but there are no restrictions on the form of this distribution. After the normalization, the phenotype samples form clusters of maximum width less than some value $\delta$, with $0 < \delta < 1$. Now, if we assume that $\delta$ is an input parameter given by the user, we define any cluster of maximum width less than $\delta$ to be a ‘$\delta$-valid pattern’. In Figure 1b, two patterns are formed, one at lower expression levels with three samples and another at higher expression levels with five samples. The number of samples in the pattern is called the ‘support’ of the pattern. Notice that we must ensure that the support of the pattern is large enough such that the pattern is unlikely to occur from sampling a uniform distribution. This issue of randomly occurring patterns will be discussed later on.

**Gene expression patterns**

Having defined a method for gene expression normalization and identification of single gene patterns, we proceed to extend the concept of $\delta$-valid patterns to multiple genes. We shall also define a few terms used throughout this paper, following the notation introduced in Califano et al. (2000).

The data for the phenotype under study contains $N_p$ microarray samples with each microarray measuring the expression level of $N_g$ genes. Thus, the phenotype dataset is conveniently represented by an $N_e \times N_g$ gene expression matrix $V = [v_{ij}]$, where $i$ is the microarray or experiment index and $j$ is the gene index. Each gene is normalized to a value in the interval $[0, 1]$ as described in the previous section. We introduce the
Fig. 1. (a) Gene expression normalization. The normalization uses an estimate of the cumulative probability density over observed control samples, \( \hat{F}_g(x) \). When the control samples are normalized by the transformation \( \mu(x) = \hat{F}_g(x) \), they become uniformly distributed in \([0, 1]\). (b) When the normalization, as given by the transformation \( \mu(x) = \hat{F}_g(x) \), is applied to samples that are not in the control set, the normalized values will tend to deviate from a uniform distribution. Only if the new observations are distributed as the control set will their normalized expressions be uniform. The figure depicts the case when the new samples follow a hypothetical density (dotted curve). When normalized, the new samples will distribute non-uniformly. If the samples fall in regions of low \( p_g(x) \) like groups A and C, they will form clusters in the interval \([0, 1]\) and some of these clusters may fit in an interval of width \( \delta \). If the samples fall in regions of high \( p_g(x) \) like group B, they will spread without forming detectable clusters.

Following definitions:

**Gene vector**: a list of \( k \) gene indexes \( G = \{g_1, \ldots, g_k\} \), with \( 1 \leq g_1 < g_2 < \cdots < g_k \leq N_g \).

**Experiment vector**: a list of \( l \) experiment indexes \( E = \{e_1, \ldots, e_l\} \), with \( 1 \leq e_1 < e_2 < \cdots < e_l \leq N_e \).

**\( \delta \)-valid pattern**: a gene vector \( G \) and an experiment vector \( E \) uniquely define an \( l \times k \) submatrix of the phenotype data matrix \( V \). Such a sub-matrix is a \( \delta \)-valid pattern if the entries in each column (genes) are contained in an interval of size up to \( \delta \).

**\( \delta \)-valid maximal pattern**: a \( \delta \)-valid pattern that cannot be extended to another \( \delta \)-valid pattern by adding genes without reducing the dimension of the experiment vector, or by adding experiments without reducing the dimension of the gene vector.

Figure 2 shows an example of a maximal pattern in a normalized gene expression matrix. A \( \delta \)-valid maximal pattern will be simply called a ‘pattern’ from now on. The length of the experiment vector \( l \) is called the support of the pattern. Observe that any sub-matrix of a pattern is also a pattern, but we are interested in \( \delta \)-valid maximal patterns that by definition cannot be completely included into another \( \delta \)-valid pattern.

Fig. 2. Example of a maximal \( \delta \)-valid pattern in a normalized gene expression matrix. In this example \( \delta = 0.1, N_e = 10, N_g = 10 \). The pattern contains three genes in its gene vector \( G = \{2, 4, 7\} \) and four experiments \( E = \{2, 3, 7, 10\} \). The pattern support is 4. The pattern is maximal because no other gene can be added that satisfies the \( \delta \) condition on \( E \) and no experiment can be added without a gene in \( G \) failing the \( \delta \) condition. Naturally, there are many more maximal patterns in this matrix besides the one shown. In bold face, are expression values that also form patterns.
Statistical significance of patterns

We are interested in discovering patterns that differentiate the phenotype set from the control set, and that are unlikely to occur by chance. In order to achieve such discrimination of patterns, we define a null hypothesis under which we can compute the probability of a given pattern to occur. Ideally, the null hypothesis would be that the genes in the phenotype set have the same joint distribution as the genes in the control set. However, the joint probability is not easily estimable with the small number of samples available in the control set. Instead, we define the null hypothesis as consisting of all genes independently distributed, with each gene distributed as $F_g(x)$. The rationale is described next. Under this null hypothesis every pattern will be assigned a $p$-value. If the $p$-value is small enough, we can assert that either the pattern genes have distributions differing from the control, or that the genes are not statistically independent. Both negations of the null hypothesis will yield a group of interesting genes. Statistical significant patterns in the majority of the cases will differentiate the phenotype from the control. However, patterns may arise that reject the null hypothesis and yet are likely to occur in the control set. We call these promiscuous patterns. These are patterns that negate the null hypothesis only because the participating genes are statistically dependent. In these cases, it could be that this dependence is also present in the control set, in which case the pattern would not be discriminating between phenotype and control. By checking whether these patterns match the control set, promiscuous patterns can be easily eliminated in a post-processing phase (Califano et al., 2002).

As explained in the normalization section, we normalize gene expressions for each gene $g$ by applying the transformation (1). Since, when we apply the normalization to the control samples we get gene expressions uniformly distributed in $[0, 1]$, in the normalized gene expression space, our null model is a set of $N_e$ independent identically distributed uniform random variables. The average number of $\delta$-valid maximal patterns with $k$ genes and support $l$ that occur in such null model can be computed by the formula (Califano et al., 2000)

$$N_{lk}(l,k,N_e,N_g,\delta) \approx \binom{N_e}{k} \binom{N_g}{l} \alpha^k \left(1 - \alpha\right)^{N_e-k} \times \left[1 - (1 + 1/l)^k \delta^k\right]^{N_g-l},$$

(2)

where

$$\alpha = l\delta(l-1) - (l-1)\delta^l.$$  

(3)

A calculation of a similar nature, but with $l = N_e$ and with binary data (i.e. overexpressing and underexpressing genes) has also been considered in Wahde et al. (2002).

We have observed in numerical experiments that the distribution of the number of $\delta$-valid maximal patterns with a fixed support and number of genes is well approximated by Poisson distribution. Assuming that to be the case, the $p$-value of a $\delta$-valid maximal pattern with $k$ genes that holds over $l$ experiments (taken to be the probability that one or more such patterns occur by chance) is

$$p = 1 - e^{-N_e}. \quad (4)$$

A pattern is considered statistically significant if its $p$-value is less than a pre-defined threshold parameter $r$. In addition, we can also compute the conditional significance of one pattern given another overlapping pattern (data not shown). In this way patterns that are likely to occur because they are a variation of another more significant pattern are discarded. The resulting set of patterns is said to be ‘non-redundant’.

Having defined the notion of $\delta$-valid maximal patterns, we now turn our attention to the algorithmic problem of detecting all these patterns from a gene expression matrix. Readers not interested in these algorithmic considerations can skip to the case study below showing a specific example of the use of Genes@Work for gene selection in cancer data.

ALGORITHM

The AddG pattern discovery algorithm

Given a normalized gene expression matrix of $N_e$ rows (microarrays) and $N_g$ columns (genes), any subset of the rows and any subset of the columns could form a pattern whenever such combination satisfies the $\delta$ condition. There are $(2^{N_e} - 1)(2^{N_g} - 1)$ distinct submatrices that might define a pattern. An enumeration of all such submatrices is impractical given that typically $N_e = 100$ and $N_g = 10\,000$. Thus, the solution space has to be searched without full enumeration. Some existing association rule discovery algorithms such as the ‘Apriori’ approach (Agrawal and Srikant, 1994) are not adequate for this domain because although they proved successful in discovering associations in large databases of sales transactions, good performance was only found for small itemsets of about six items (Agrawal and Srikant, 1994). Efficient searches will exploit the property that any sub-matrix of a pattern is also a pattern by growing smaller patterns into maximal ones, thus reusing computation and pruning the search space in a similar fashion to that of dynamic programming algorithms (Cormen and Leiserson, 1990).

The AddG algorithm is an efficient approach to finding the maximal patterns in the data. The input parameters for the algorithm are the maximum expression interval ($\delta$), the minimum support ($l_{\min}$), and the minimum number of genes ($k_{\min}$). The maximum $p$-value or threshold $r$ required for a pattern is applied to each maximal pattern at the end of the pattern search, hence pattern discovery performance does not depend on the parameter $r$ for the algorithm described.

The algorithm starts by searching single gene elementary patterns that are maximal in the number of experiments and contain at least $l_{\min}$ experiments. Any arbitrary order for the genes is fixed and each gene is considered in turn.
Fig. 3. Constructing the maximal pattern of Figure 2 by the AddG algorithm. Starting with the maximal single gene elementary patterns π₂, π₄ and π₇, we can obtain the maximal pattern of Figure 2 by the intersection of the experiment sets from the elementary patterns and the union of the corresponding genes.

The elementary pattern search can be done efficiently by sorting the normalized gene expression of each gene and then looking for a window of experiments where the δ condition holds. The elementary patterns will be grown by adding genes as described below. As genes are added, maintaining the δ condition valid over the common set of experiments in the pattern will require removing experiments from the grown pattern (Fig. 3). The growth continues until no more genes are available or until the pattern fails to meet a minimum required number of experiments \(l_{\text{min}}\). In Figure 3, the patterns π₂, π₄ and π₇, are examples of single gene elementary patterns for the data in Figure 2.

The elementary patterns constitute the basic building blocks for the resulting maximal patterns. The order in which the genes are considered for the elementary pattern search together with the order in which the elementary patterns are generated for each gene, define a global order for all elementary patterns. All elementary patterns are stored in an array that maintains this generation order.

Each elementary pattern defines a subset of the experiments called an 'experiment comb'. The experiment comb can be represented efficiently as a bit vector of length \(N_e\) with a one (respect zero) indicating that the experiment is present (respect absent) in the elementary pattern. For reasons that will be clear later, we first determine a subset of the elementary patterns whose experiment comb is not included in any previous experiment comb. Here, the order of the combs is simply given by the generation order of elementary patterns. If we ordered the elementary patterns from left to right, we can call this subset of elementary patterns as the 'left-maximal' elementary patterns. The experiment comb of a left-maximal elementary pattern is not included in the experiment comb of any previous (i.e. located to the left) elementary pattern.

Now we are ready to describe the main loops of the algorithm. The outermost loop goes over all the genes; we call the current gene in this loop the ‘source’ gene. The task of the outermost loop’s body is to find all maximal patterns that start with the source gene. The first step is to retrieve the elementary patterns present at the source gene. We call these elementary patterns the ‘source seed patterns’. The next outermost loop is started to consider each source seed pattern in turn.

For each source seed pattern, a new loop is started over the genes that follow the source gene according to the selected gene order. We call the current gene in this new loop the ‘target’ gene. Each target gene contains elementary patterns called ‘target seed patterns’. A combine operation between patterns is defined such that it produces a new pattern whose experiment comb is the intersection of the experiment combs of the argument patterns and whose genes are the union of the genes in the argument patterns. In Figure 3 we illustrate the pattern combination operation which we also refer to as pattern intersection. The intersection of the source seed pattern with all the target seed patterns is computed. Each intersection is a new two-gene pattern containing the gene of the source seed and the gene of the target seed. The experiment comb of the intersection is simply the bit-by-bit logical AND operation of the experiment combs of the source seed and the target seed. The processing can be described by the following Java-like pseudocode:

```java
for(int sg=1;sg<=N_g;sg++)
{ //Outermost loop over source gene, sg.
  PatternList sourceGeneSeeds = getElementaryPatternsForGene(sg);
  for(int i=1;i<=sourceGeneSeeds.length;i++)
  { //Elementary source pattern,
    Pattern elemSourcePattern = sourceGeneSeeds[i];
    //Initialize a list of patterns.
    PatternList twoGenePatterns = new PatternList();
    for(int tg=sg+1;tg<=N_g;tg++)
    { //Inner loop over target gene, tg.
      PatternList targetGeneSeeds = getElementaryPatternsForGene(tg);
      //Combinatorial operation...
    }
  }
}
```
for(int j=1;j<=targetGeneSeeds.length;j++)
{
    Pattern elemTargetPattern = targetGeneSeeds[j];
    Pattern newPattern = intersect(elemSourcePattern, elemTargetPattern);
    if (newPattern.experimentCount>=lmin &&isExpCombNew(newPattern))
    {
        twoGenePatterns.append(newPattern);
    }
}

//Combine two-gene patterns into
//n-gene patterns.
growPatterns(twoGenePatterns);
}

The new two-gene pattern is kept only if its support is greater than or equal to \( l_{\text{min}} \). In addition, its experiment comb is checked against all 'left-maximal' experiment combs from the genes located before the current source gene with the call to isExpCombNew(newPattern). If any previous comb contains the experiment comb of the new two-gene pattern, the two-gene pattern is discarded. As we shall see, such pruning is justified and saves a search that cannot generate any new maximal patterns.

The next step is to combine the two-gene patterns into patterns with more genes until maximality in number of genes is reached. The procedure is described conceptually in Figure 4 and with more detail in the following pseudocode:

```plaintext
function growPatterns(PatternList twoGenePatterns)
{
    //Function growPatterns: Combine two-gene patterns into n-gene patterns.
    Stack stack = new Stack();
    //Use a stack to backtrack pattern search.
    stack.push(twoGenePatterns);
    //Stack holds pending pattern lists, init.

    while(!stack.empty())
    {
        PatternList prevLevelPatterns = stack.pop();
        //Level (n-1)-gene patterns.
        while(prevLevelPatterns!=[]) {
            Pattern sourcePattern = LIST_HEAD(prevLevelPatterns);
            PatternList prevLevelPatterns = LIST_TAIL(prevLevelPatterns);
            (B) if (!NOT(sourcePattern.isKept())) continue;
            (H) if (!NOT(isExpCombNew2(sourcePattern)))
            //Search pruning step.
            continue;
            PatternList nextLevelPatterns = new PatternList();
            for(int j=1; j<=prevLevelPatterns.length; j++)
            {
                Pattern targetPattern = prevLevelPatterns[j];
                (C) if (!NOT(targetPattern.isKept())) continue;
                Pattern newPattern = intersect(sourcePattern, targetPattern);
                (F) if (newPattern.experimentCount>=lmin &&isExpCombNew(newPattern))
                {
                    //Search pruning step.
                    //Verify maximality of source and target patterns.
                    if (sourcePattern.experimentCount==newPattern.experimentCount)
                    (D) sourcePattern.setKeep(false);
                    if (targetPattern.experimentCount==newPattern.experimentCount)
                    (A) targetPattern.setKeep(false);
                    //Save new level n-gene pattern.
                    nextLevelPatterns.append(newPattern);
                }
            }
            //If source pattern could not be extended
            //at same support,
            //it must be maximal, add to solution.
            if (sourcePattern.isKept())
            (E) maximalPatterns.append(sourcePattern);
            if (nextLevelPatterns!=[]) {
                stack.push(prevLevelPatterns);
                //Backtrack pointer, process later.
                stack.push(nextLevelPatterns);
                break; //Break out to process
                //newest list from top of
                //stack.
            }
        }
    }
}
```

In this procedure, we always intersect a source pattern and a target pattern with the same number of genes and that differ...
The pattern search/generation process. Starting with a list of \(t\) two-gene (level 2) patterns, we take the first pattern in the list (head) and intersect with the remainder patterns in the list (tail). \(g_i\) is the current source gene, \(g_1\)–\(g_t\) are the target genes. This intersection produces new patterns that have an additional gene and a possibly new experiment comb. The old list’s tail is pushed into the pattern list stack and the process is repeated on the new list. At some point, all pattern lists in the stack have been processed (the pattern list stack is empty) the pattern search is complete. Note that although in the figure the height of each pattern is drawn equally, each pattern can have very different experiment combs.

only in their last gene (except if a target gene had multiple seed patterns, but we omit details of this case for clarity). The resulting pattern contains the union of the genes in both patterns intersected. We say that the source and target are in the previous level, and the intersection pattern in the next level, where the level is given by the number of genes in the patterns. In comparison with the Apriori algorithm that requires all sub-patterns, but has more genes and when expanded includes the experiment comb of one of these elementary patterns, we take the first pattern in the list (head) and intersect with the remainder patterns in the list (tail). \(g_i\) is the current source gene, \(g_1\)–\(g_t\) are the target genes. This intersection produces new patterns that have an additional gene and a possibly new experiment comb. The old list’s tail is pushed into the pattern list stack and the process is repeated on the new list. At some point, all pattern lists in the stack have been processed (the pattern list stack is empty) the pattern search is complete. Note that although in the figure the height of each pattern is drawn equally, each pattern can have very different experiment combs.

In comparison with the Apriori algorithm that requires all sub-patterns, but has more genes and when expanded includes the experiment comb of one of these elementary patterns, we take the first pattern in the list (head) and intersect with the remainder patterns in the list (tail). \(g_i\) is the current source gene, \(g_1\)–\(g_t\) are the target genes. This intersection produces new patterns that have an additional gene and a possibly new experiment comb. The old list’s tail is pushed into the pattern list stack and the process is repeated on the new list. At some point, all pattern lists in the stack have been processed (the pattern list stack is empty) the pattern search is complete. Note that although in the figure the height of each pattern is drawn equally, each pattern can have very different experiment combs.

...
is not shown to scale. For all runs, a value of $\delta = 0.1$ is used. The gene expression is independent, identical and uniformly distributed in $[0, 1]$ (see right insert) except for the gene and experiment number of maximal patterns is strongly dependent on the input scenario for the algorithm. This indicates that there will be cases in which the search for patterns with both low support and few genes can become computationally costly. However, we can control the size of the solution by asking for patterns with high support or with many genes or both. In practice, the user must tune the algorithm parameters so that only significant and mostly non-redundant patterns are reported. For example, by demanding that only conditionally significant patterns are considered (i.e. patterns that cannot be explained in terms of other patterns), only a handful of all the maximal patterns will be reported. These patterns are variations of the three planted patterns shown in Figure 5. For the purpose of assessing performance, we count all maximal patterns regardless of their statistical significance or redundancy.

The running times versus the solution size for three values of $N_e$ are shown in Figure 7. The differences in the curves with different $N_e$ is due to the fact that for lower $N_e$, we are finding patterns with more genes (and lower support) which requires the algorithm AddG to make a deeper search when growing the patterns. For each $N_e$, the running times increase slightly
faster than linearly with the size of the solution. A polynomial fit to the data of Figure 7 reveals a small quadratic term (for \( N_{p} = 50 \), however, the quadratic term was negligible). This suggests that there could be some room for improvements to the AddG algorithm given that the optimal runtime is linear in the output size. However, this sub-optimal behavior has not posed serious problems at the time of analyzing real datasets, as will be exemplified in the following section.

**Gene selection using Genes@Work: a case study**

Genes@Work has recently been used to explore gene expression data in a number of bio-medical investigations (Jelinek et al., 2003; Klein et al., 2001, 2003; Kuppers et al., 2003). However, we have not performed a comparative study with other methods, or presented distinct evidence of the relationship between the statistical significance of our patterns and the biological relevance of the resulting genes. We will do this next.

In this section, we describe the application of pattern discovery to the identification of interesting genes in lymphoma data. In particular, we seek genes that differentiate between two types of lymphomas, diffuse large B-cell lymphomas (DLBCLs) and follicular lymphomas (FLs). Gene expression data for both types of lymphomas has been analyzed in Shipp et al. (2002). The analysis method in Shipp et al. (2002) is based on selecting a number of genes with the highest signal to noise ratio (SNR) score. The SNR score is defined as

\[
\text{SNR} = \frac{\mu_0 - \mu_1}{\sigma_0 + \sigma_1}
\]

where, for each gene, \( \mu_0 \) and \( \sigma_0 \) are the mean and SD respectively over one group of samples, and \( \mu_1 \) and \( \sigma_1 \) are the mean and SD respectively over the other group of samples. The 50 largest and positive scoring genes (genes more expressed in DLBCL than in FL) and the 50 largest and negative scoring genes (genes more expressed in FL than in DLBCL) were selected, for a total of 100 genes. Each of these 100 genes appear to have a statistically significant pushing power (SNR) of over 1% when its SNR was assessed against the other group of samples. The 50 largest and negative scoring genes (genes more expressed in DLBCL than in FL) were selected, for a total of 100 genes.

There are 41 genes that are reported by both SNR method of Shipp et al. (2002) and our Genes@Work pattern discovery method. Genes@Work chose 59 genes that SNR detection did not choose and SNR chose 57 genes that Genes@Work did not choose. The fact that the 100 most significant genes reported by both methods are different is the result of the type of signal sought by each method. In Genes@Work, a gene must associate with other genes through a pattern to be reported, on the other hand, when selecting by SNR each gene is considered in isolation.

The gene expression patterns discovered by Genes@Work need not be composed of genes that are either over or under expressed in the phenotype set versus the control set. Indeed, the requirement of a pattern is that its genes express at a consistent level in the phenotype set, but such level of expression could lay anywhere within the range of the control set. However, if the parameter \( \delta \) is sufficiently small, only over or under expressed genes in a subset of samples will tend to participate in the formation of patterns. In this way, the parameter \( \delta \) controls the degree to which the selected pattern genes produce a neat separation of phenotype and control sets into overexpressing and under expressing genes. For the \( \delta \) value considered in this example, the selected genes do segregate into groups that over and under express in phenotype versus control.

The usual method used in the literature for the validation of a set of markers consists of performing a classification check on the same dataset on which the markers were found (using approaches such as leave-one-out cross validation), or by choosing a somehow stringent threshold for the
Fig. 8. (a) Verification of gene signatures with an independent dataset. WI are the Whitehead Institute samples, CU are the Columbia University samples. The left dendrogram plots genes reported only by the SNR method. The right dendrogram plots genes reported by both the SNR method and the Genes@Work based method. (b) Verification of gene signatures with an independent dataset. The dendrogram plots genes reported only by Genes@Work and not by the SNR method.
statistical significance when choosing the markers. In Shipp et al. (2002), e.g. 30 of the 100 markers were verified as informative by using the gene voting scheme introduced in Golub et al. (1999). As stated above, all the 100 markers also passed a test of statistical significance. In our case, we could state that the statistical significance of the patterns found by Genes@Work was stringent given that the $p$-value of the least significant pattern was $10^{-10}$. However, statistical significance need not mean biological relevance.

To demonstrate further that the extra 59 genes reported by Genes@Work are not the result of artifacts, but contain reproducible information about the phenotypes DLBCL and FL, we obtained an independent dataset on the same types of lymphomas. This independent dataset, from Riccardo DallaFavera’s laboratory at Columbia University (CU), contains 14 DLBCL samples and 7 FL samples. [This data had previously been used in Klein et al. (2001) and Kuppers et al. (2003).] We plot the data from the Whitehead Institute (WI) used in Shipp et al. (2002) and CU side-by-side in Figure 8a and b, so we can visually inspect if a gene reproduces its behavior across the independent dataset. Figure 8a considers two groups of genes, those reported only by the SNR method and those reported by both SNR and Genes@Work. Figure 8b shows those genes only reported by Genes@Work. Figure 8b shows that the genes reported only by Genes@Work maintain their signature in a dataset independent from the dataset used for analysis. Each gene exhibits a relatively homogeneous profile inside the DLBCL and FL phenotypes because each gene is constrained to form a pattern covering at least 67% of samples, see setting of $l_{\text{min}}$ parameter above. A reasonable formalization of the visual consistency observed in Figure 8b is necessary. We will say that a gene is consistent if the sign of the difference of the average expression in DLBCL and FL is the same in both datasets. The $p$-value for the number of consistent genes is defined as the probability that the same number of genes found to be consistent in the two datasets were to be found consistent if the genes were independent random variables with equal probability of being consistent or not (i.e. the null hypothesis assumes that in the test dataset each gene is independent and has equal probability of overexpressing or underexpressing in DLBCL versus FL). Of the 59 genes found uniquely by Genes@Work in the Whitehead dataset, 58 genes were consistently expressed in the Columbia dataset, which corresponds to a consistency $p$-value of $10^{-16}$. We conclude that the new genes identified by Genes@Work are highly reproducible in the independent dataset.

Finally, the question remains as to whether the 59 genes specific of the Genes@Work method would have been found by the SNR method if we relaxed the stringency of the latter to allow for a few more genes than the 100 reported in Shipp et al. (2002). To account for 75% of the 59 genes, the stringency of the SNR method has to be relaxed to allow for more than 700 extra genes, indicating that many of these Genes@Work specific genes are of low statistical significance according to the SNR method. In other words, Genes@Work rescues as true positives genes that the SNR method deemed as negatives.

**SUMMARY AND CONCLUSIONS**

In this paper, we have described the Genes@Work approach for pattern discovery in gene expression data. At the core of Genes@Work, the AddG algorithm shows good performance and scalability under most conditions of interest. As an application of Genes@Work, we showed an example of how patterns can be used to identify genes that express differentially in two lymphomas, DLBCL and FL, as seen in the case study above. We found a subset of 59 genes that were specifically picked by pattern discovery in Genes@Work but not by the SNR method. Of the 59 genes, all but one showed the same profile of expression in the independent dataset, thereby validating the differential expression of those genes in DLBCL and FL.

Because Genes@Work probes the data with questions not usually asked by other methods, our approach can be complementary to other analytical methods. Let us emphasize that in no way do we wish to imply that Genes@Work is a better method than others under all circumstances. Our results suggest that the richness of the gene expression data demands that more than one method be used in its analysis. In this way, we believe that Genes@Work should share with other single gene and multivariate methods in the gene expression analyst’s tool kit.

**ACKNOWLEDGEMENTS**

We are deeply indebted to Andrea Califano who participated in the early stages of the development of Genes@Work. We are also grateful to Riccardo DallaFavera and Ulf Klein for their encouragement and suggestions in the design of Genes@Work, and for facilitating the test data used in this paper.

**REFERENCES**


