Supplemental File 2: Module generation and pseudocode

Each of the preprocessed microarray datasets was clustered in parallel using Euclidean distance and the Hartigan's K-Means clustering algorithm, a hybrid of hierarchical and K-Means clustering algorithms. We made an additional modification to the algorithm. If at any k the algorithm creates a cluster whose members' average Pearson correlation to the mean cluster vector is less than 0.3, the cluster is deleted, and the algorithm begins again at k-1. The 'ideal' number of clusters (k) for each dataset was determined within a range of k=1 to 100 by means of the jump statistic.

Taking the sixteen sets of clusters as input, we constructed a weighted co-cluster graph, a probe by probe matrix where the value of each cell is set to a 16-bit bitmask (each bit corresponding to a dataset) indicating in which datasets the probes were co-clustered. The bit is set to 1 if the probe pair co-clusters in the corresponding dataset, and to 0 if not. Therefore, the number of times the probes co-clustered (the weight) is equivalent to the number of bits in the mask that are set to 1. At this point, the goal is to extract sets of probes that are most frequently clustered together in the same datasets, proceeding from the most stringent requirements to the least. To accomplish this, we employ an iterative algorithm. To begin, the maximum clique threshold is initialized to the number of input cluster sets, the paraclique threshold (pt) is calculated, and a minimum seed size is chosen (we used fifteen). The outer loop begins by creating an unweighted graph through application of the maximum clique threshold (mct) to the weighted co-cluster graph such that a probe pair, or edge, is connected in the unweighted graph if and only if the corresponding weight in the co-cluster graph equals or exceeds this threshold.

We then begin the inner loop. The first step is to isolate the largest set of probes such that all probes in the set are completely connected in the unweighted graph. In graph theoretic terms, the probes form a maximum clique. We impose an additional constraint that all probes in the set must co-cluster in the same datasets at least mct times by taking the intersection of the bitmasks for every edge in the clique and requiring that the result contain at least mct bits set to 1. This new bitmask is the common co-cluster bitmask for the clique.

If the size of the clique is smaller than the minimum seed size, we escape from the inner loop, reduce mct by one, and return to the beginning of the outer loop. Otherwise, it becomes the seed for a module. To allow for the inevitable clustering inaccuracies, we then employ the paraclique algorithm. We revisit the co-cluster graph and add to the seed any probe that is found to co-cluster with at least ninety percent of the seed's members in at least pt of the datasets represented in the clique's common co-cluster bitmask. This final probe set is a module. It is removed from both graphs and named in accordance with the iterations in which it was found (i.e. a module extracted in the first iteration of the outer loop and the second iteration of the inner loop is designated M1.2). The inner loop then begins again with the reduced graphs. Pseudocode is shown below:

```
Integer nLastQuartile = 4;
Integer nMaxRelaxtion = m_nNumDatasets / 3;
Integer nRelaxtionIncrement = Math.max(1, (nMaxRelaxtion / 3));
Integer nRelaxtion = nMaxRelaxtion;
for (int nCliqueThreshold = numberOfDatasets; nCliqueThreshold >= 1; nCliqueThreshold--)
{
	Integer nQuartile = ((nThreshold * 100) / m_nNumDatasets) / 25;
	if (nQuartile.equals(nLastQuartile) == false)
	{
		if (nQuartile <= 2)
		{
			nRelaxtion = Math.max(0, nRelaxtion - nRelaxtionIncrement);
		}
```

```
        nLastQuartile = nQuartile;
}
Integer nParacliqueThreshold = nThreshold - nRelaxtion;

do
{
        maximumClique = find maximum clique w co-clustering weight >= nCliqueThreshold
        if (size of maximumClique > 15)
        {
                paraclique = find paraclique in graph
                remove maximumClique and paraclique from graph
        }
}   while (maximumClique is found)
```