Exploring similarities across high-dimensional datasets *

Karlton Sequeira and Mohammed Zaki
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180

Abstract

Very often, related data may be collected by a number of sources, which may be unable to share their entire datasets for reasons like confidentiality agreements, dataset size, etc. However, these sources may be willing to share a condensed model of their datasets. If some substructure of the condensed models of such datasets, from different sources are found to be *unusually similar*, policies successfully applied to one may be successfully applied to the others. In this paper, we propose a framework for constructing condensed models of datasets and algorithms to find similar substructure in these models. The algorithms are based on the tensor product and Gibbs sampling. We test our framework on synthetic datasets and compare our algorithms with an existing one. Finally, we apply it to two time-course microarray datasets of different dimensionality. The results are statistically, more interesting than results obtained from independent analysis of the datasets.

1 Introduction

Often, data may be collected by a number of sources. These sources may be geographically far apart. There are a number of disadvantages in transferring the datasets from their source to a central location for processing. These include less reliability, security, higher computational and storage requirements, etc. It may be preferable to share condensed models of the datasets. Similarly, for reasons like confidentiality agreements, etc., it may be required to use condensed models of datasets, which obfuscate individual details, while conveying structural information about the datasets. Lastly, the datasets may have slightly different schema or transformations like rotations, with respect to each other. This may preclude simply appending the datasets to each other and processing them.

If *unusually similar* substructure can be detected from the condensed models of some of the datasets, then policies successfully applied to one, may be

*This work was supported by NSF Grant EIA-0103708 under the KD-D program, NSF CAREER Award IIS-0092978, and DOE Early Career PI Award DE-FG02-02ER25538.

successfully applied to the others. For example, two consumer markets (A and B) differing in geography, economy, political orientation or some other way may have some *unusually similar* consumer profiles. This may prompt sales managers in B to use successful sales strategies employed by sales managers in A for consumer profiles in which they are *unusually similar*. Also, profiles which are *unusually dissimilar* to any of those in the other graph are particularly interesting. The latter is analogous to the problem of finding contrast sets [2]. Additionally, determining similarities and dissimilarities between snapshots of a dataset taken over multiple time intervals can help in identifying how the dataset characteristics evolve over time [9].

A dataset may be a set of points drawn in possibly different proportions, from a mixture of unknown, multivariate and perhaps, non-parametric distributions. A significant number of the points may be noisy. There may be missing values as well. We currently assume that the dataset may belong to non-identical attribute spaces, which are mixtures of nominal and continuous variables. The datasets may be subject to translational, rotational and scaling transformations as well. High dimensional datasets are inherently sparse. It has been shown that under certain reasonable assumptions on the data distribution, the ratio of the distances of the nearest and farthest neighbors to a given target is almost 1 for a variety of distance functions and data distributions [3]. Hence, traditional distance metrics which treat every dimension with equal importance have little meaning. Algorithms using such dissimilarity measures as a building block for application to high-dimensional datasets, may produce meaningless results due to this lack of contrast.

In this paper, we explore similarities across datasets using a two step solution:

1. Constructing a condensed model of the dataset. This involves finding the components of the model and relationships between these components. In our case, the components are subspaces (see Definition 1). The condensed model is a weighted graph where the vertices correspond to subspaces and the weighted edges to relationships between the subspaces. A condensed model allows
    - sharing of dataset summaries.
    - noise and outlier removal.
    - normalization and dataset scaling.
2. Identifying similarities between the condensed models.

In previous work [19], we have shown algorithms to find components of the model. In this paper, we make the following contributions:

- We propose two kinds of similarity measures for subspaces(components). The first kind is projection-based, i.e., it uses the similarity of the projections of the subspaces. The other is support-based, i.e., it uses the number of points shared by the subspaces.
- We provide algorithms for identifying *unusually similar* substructure from the condensed models corresponding to datasets with possibly differing schema.

- We test our framework with synthetic datasets and apply it to finding similar substructure in models constructed from yeast cell cycle microarray datasets. These datasets have different dimensionality and are generated by employing differing experimental methods. Inferences from the similar substructure are found to be biologically meaningful. Further, they reveal information, which remains unknown under independent dataset analysis.

1.1 Preliminaries

Consider dataset D_A having d_A dimensions. If $S_{A,i}$ is the domain of the ith dimension, then $S_A = S_{A,1} \times S_{A,2} \times \ldots \times S_{A,d_A}$ is the high-dimensional space for D_A, where $D_A = \{x_i | i \in [1,m], x_i \in S_A\}$. Similarly, $D_B = \{y_i | i \in [1,n], y_i \in S_B\}$. If the range $S_{A,i}$ of each dimension is divided into ξ equi-width intervals, then S_A has a grid superimposed over it. Accordingly, we have the following definition:

Definition 1 *A subspace is a grid-aligned*[1] *hyperrectangle* $[l_1, h_1] \times [l_2, h_2] \times \ldots \times [l_d, h_d]$*, where* $\forall i \in [1,d], [l_i, h_i] \subseteq S_{A,i}$.

If $[l_i, h_i] \subset S_{A,i}$, the subspace is said to be *constrained* in dimension i, i.e., the subspace does not span the entire domain of the dimension i. $l_i = (aS_{A,i})/\xi$, and $h_i = (bS_{A,i})/\xi$, where a, b are non-negative integers, and $a < b \leq \xi$.
A subspace, which is constrained in all the dimensions to a single interval, i.e., $b - a = 1$, is referred to as a *grid cell*.
If there are $|V_A|$ subspaces internal to D_A, the inter relationships between these subspaces are expressed by a $|V_A| \times |V_A|$ matrix $w_A : S_A \times S_A \rightarrow \Re$.

Definition 2 *Let* $A = (a_{i,j})_{1 \leq i,j \leq m,n}$, $B = (b_{kl})_{1 \leq k,l \leq p,q}$.
If $m = n$, $Tr[A] = \sum_{i \in [1,m]} a_{i,i}$ *is the trace of* A
A^T *refers to the transpose of* A.
$||A||_F = (\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|^2)^{1/2}$ *is the Frobenius norm of* A.
ones(m,n) *returns a* $m \times n$ *matrix containing all ones.*
The tensor product[2] *of* A *and* B *is a* $mp \times nq$ *matrix, and is defined as*

$$A \otimes B = \begin{pmatrix} a_{1,1}B & a_{1,2}B & \ldots & a_{2,n}B \\ a_{2,1}B & a_{2,2}B & \ldots & a_{2,n}B \\ a_{m,1}B & a_{m,2}B & \ldots & a_{m,n}B \end{pmatrix}$$

For normal $n \times n$ matrix X, on eigendecomposition, $X = U_X D_X U_X^T$, where U_X is a unitary matrix containing the eigenvectors of X and D_X is a diagonal matrix containing the eigenvalues of X. $\lambda_{X,i}$ denotes the ith eigenvalue, where $\forall i \in [1, n-1], \lambda_{X_i} \geq \lambda_{X,i+1}$ and $U_{X,i}$ denotes the eigenvector corresponding to $\lambda_{X,i}$. If $\lambda_{X,1} > \lambda_{X,2}$, $\lambda_{X,1}$ and $U_{X,1}$ are called the dominant eigenvalue and dominant eigenvector respectively.

If $S = [\mathbf{s_1}\ \mathbf{s_2}\ \ldots]$ where $\mathbf{s_1}, \mathbf{s_2}, \ldots$ are column vectors, then $vec(S)$ creates a column vector by stacking its column vectors one below the other, so that $vec(S) = [\mathbf{s_1^T}\ \mathbf{s_2^T}\ \ldots]^T$.

[1]grid-aligned subspaces are essential for interpretability
[2]Also called the matrix direct product or Kronecker product

Let P be the function, which takes as argument a mapping $f : V_A \rightarrow V_B$, and returns a permutation matrix[3], i.e., a $|V_A| \times |V_B|$ matrix, such that

$$P_f(u, v) = \begin{cases} 1 & \text{if } f(u) = v \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

If f is a one-to-one mapping, then the rows and columns of P are orthogonal to each other and $PP^T = I$. If $|V_A| = |V_B|$, P is an orthogonal matrix. We want f which minimizes the associated error function err, which we define as

$$err(f|w_A, w_B) = ||w_A - P_f w_B P_f^T||_F \tag{2}$$

A mapping f from a subset of subspaces corresponding to w_A to a subset corresponding to w_B is **unusually similar**, *if the probability of finding another mapping f' between these subsets, by MonteCarlo sampling as described in Section 4.4, such that $err(f|w_A, w_B) > err(f'|w_A, w_B)$, is very low.*

1.2 Example

D_A	d_1	d_2	d_3	d_4	d_5
p_1	915	561	866	657	661
p_2	965	575	534	860	365
p_3	217	506	121	452	303
p_4	758	512	357	423	289
p_5	276	531	327	418	335
p_6	268	520	351	348	454
p_7	239	514	369	301	451
p_8	237	510	377	650	472
p_9	33	118	144	388	280
D_B	d'_1	d'_2	d'_3	d'_4	d'_5
p'_1	889	710	591	564	679
p'_2	854	189	641	564	666
p'_3	553	869	449	612	199
p'_4	779	690	203	598	872
p'_5	88	453	965	541	324
p'_6	391	436	193	578	301
p'_7	574	450	220	588	270
p'_8	805	60	803	525	152

Table 1: Original datasets

D'_A	d_1	d_2	d_3	d_4	d_5
g_1	9	5	8	6	6
g_2	9	5	5	8	3
g_3	2	5	1	4	3
g_4	7	5	3	4	2
g_5	2	5	3	4	3
g_6	2	5	3	3	4
g_7	2	5	3	3	4
g_8	2	5	3	6	4
g_9	3	1	1	3	2
D'_B	d'_1	d'_2	d'_3	d'_4	d'_5
g'_1	8	7	5	5	6
g'_2	8	1	6	5	6
g'_3	5	8	4	6	1
g'_4	7	6	2	5	8
g'_5	8	4	9	5	3
g'_6	3	4	1	5	3
g'_7	5	4	2	5	2
g'_8	8	6	8	5	1

Table 2: Discretized datasets

Let D_A and D_B be two datasets as shown in Table 1, with domains [0,1000) for each dimension. $D_A(p_1, d_1)$ refers to row p_1, column d_1 of dataset D_A. If we discretize the domain of each dimension into 10 intervals, i.e., $\xi = 10$. Then the grid cells surrounding the points in D_A, D_B yield the datasets D'_A, D'_B in Table 2. For example, $D_A(p_1, d_1) = 915$. Therefore, $D'_A(g_1, d_1) = \lfloor \frac{915}{1000} \times \xi \rfloor = 9$.

[3]Typically, a permutation matrix is a square matrix

Thus, p_1 is constrained to the last interval in dimension d_1, i.e., [900,1000). We then run a subspace mining algorithm (e.g., SCHISM[19], CLIQUE [1]) on each of the discretized datasets independently and find two sets of subspaces S and S' corresponding to D_A and D_B respectively, as shown in Table 3. -1 implies that the dimension is unconstrained. $\mathrm{S}(c_1, d_2) = 5$ means that the subspace c_1 in the set S of subspaces, is constrained to interval 5 in dimension d_2, i.e. [500,600). Subspaces may be constrained to more than one interval in a dimension.

S	d_1	d_2	d_3	d_4	d_5
c_1	-1	5	-1	-1	-1
c_2	-1	5	-1	4	3
c_3	2	5	3	-1	4

S'	d'_1	d'_2	d'_3	d'_4	d'_5
c'_1	-1	-1	-1	5	-1
c'_2	-1	4	-1	5	2

Table 3: Two sets of subspaces

Typically, subspace mining algorithms also partition the dataset based on the subspaces it finds. Let us assume that the subspace mining algorithm assigns p_1, p_2 to c_1, p_3, p_4, p_5 to c_2, p_6, p_7, p_8 to c_3 and labels p_9 as noise. Similarly, it assigns p'_1, p'_2, p'_3, p'_4 to c'_1, p'_5, p'_6, p'_7 to c'_2 and labels p'_8 as noise.

Given such subspaces and the points assigned to them, we wish to construct condensed models of the datasets, which can be used to discover structurally similar subspaces across the two datasets without having access to the datasets or their schema. For example, in Table 3, if d_2 corresponds to d'_4 and d_4 corresponds to d'_2, then c_1 and c'_1 are both constrained in the same dimension and to the same interval, i.e., [500,600). Also, c_2 and c'_2 are constrained in the same dimensions to similar intervals. Hence, $c_1 \sim c'_1$ and $c_2 \sim c'_2$. Thus, we wish to recover the mapping between c_1 and c'_1, and c_2 and c'_2,

2 Related Work

Our two step solution to finding *unusually similar* substructure across datasets involves:

1. Constructing a condensed model of the dataset. This involves
   - Finding components in the dataset.
   - Constructing a condensed model from the components.
2. Identifying similarities between the condensed models.

2.1 Finding components in the dataset

We find components in the dataset using a subspace mining algorithm called SCHISM [19], which finds sets of possibly overlapping subspaces, e.g., set S from dataset D_A in the example in Section 1.2. It partitions the points in the datasets using these subspaces. *Note that any other hyperrectangular subspace mining algorithm, e.g., MAFIA, CLIQUE[1], etc. may be used to find the subspaces and* *partition the dataset.* Hence, we do not delve into the details of the SCHISM algorithm.

2.2 Constructing condensed models from the components

We condense the dataset using a weighted graph, where the vertices correspond to subspaces and the weights on the edges to similarities between the subspaces. While we are unaware of much related work on similarities between subspaces, as defined in Definition 1, it is noteworthy that subspaces are also clusters. Accordingly, we review some of the existing similarity measures used for comparing clusterings.

Clusterings may be compared based on the number of point pairs, in which the two clusterings C, C' agree or disagree. Each pair of dataset points is assigned to one of four categories N_{00}, N_{01}, N_{10} and N_{11}. Pairs of points in N_{00} are assigned to distinct clusters in both C and C', those in N_{11} are assigned to the same cluster in both C and C', those in N_{01} are assigned to the same cluster in C but to distinct clusters in C', and so on. If the dataset has n points,
$N_{00} + N_{01} + N_{10} + N_{11} = n(n-1)/2$.
Accordingly there exist the Rand index

$$Rand(C, C') = \frac{N_{11} + N_{00}}{N_{11} + N_{10} + N_{01} + N_{00}}$$

the Jaccard index

$$Jaccard(C, C') = \frac{N_{11}}{N_{11} + N_{10} + N_{01}}$$

Meila [14] proposes the VI (variance of information) metric to compare clusterings.

$$VI(C, C') = H(C) + H(C') - 2I(C, C')$$

where $H(C) = \sum_{i=1}^{|C|} -p_i log(p_i)$ and
$I(C, C') = \sum_{i=1}^{|C|} \sum_{j=1}^{|C'|} p_{i,j} log(\frac{p_{i,j}}{p_i p_j})$, where $p_i = \frac{n_i}{n}$, $p_{i,j} = \frac{|C_i \cap C'_j|}{n}$, n_i being the number of points in C_i, the ith cluster in C. This implies that p_i and $p_{i,j}$ are simply the *support* of clusters C_i and $C_i \cap C_j$ respectively, according to the traditional definition of support in the data mining literature.

Thus, these clustering (dis)similarity measures use (dis)similarity in support overlap to express cluster similarity.

2.3 Identifying similarities between condensed models

Ganti *et al.* [9] compare datasets by comparing their respective models. The datasets share a common schema. A dataset may be typically modeled by a decision tree or a set of clusters or a set of frequent itemsets. The model consists of a set of pairs. Each pair consists of an "interesting region" in the dataset (called the *structural component*) and the fraction of the dataset (called the *measure component*) it accounts for. They then partition the attribute space using hyperplanes, which (as per the type of model chosen) define the leaves, clusters or frequent itemsets, induced by the models of the two datasets. Using a single scan of each dataset, they can compute the fraction of each dataset in each distinct hyperspace, resulting from the superposition of the two models of the datasets. They then compare these fractions, corresponding to different datasets but the same hyperspace, using a "difference" function and combine the resulting "deviation" using an "aggregation" function which returns a measure of the similarity of the datasets. This method does not leverage the structure present in the data and hence is susceptible to translational transformations.

Parathasarathy *et al.* [18] propose a similarity measure for comparing datasets having the same schema, based on the set of frequent itemsets in each dataset. If F_A, F_B denote the set of frequent itemsets for datasets $D_A,\ D_B$ respectively, then similarity is defined as

$$\frac{\sum_{x \in F_A \cap F_B} \max\{0, 1 - \alpha || \frac{freq(x,D_A)}{|D_A|} - \frac{freq(x,D_B)}{|D_B|} ||_1\}}{|F_A \cup F_B|}$$

where $||A||_1$ refers to sum of the magnitudes of the entries in A, the $freq(x, D_A),\ freq(x, D_B)$ are the number of occurrences of itemset x in D_A and D_B and $0 \leq \alpha \leq 1$ is a parameter, which scales the difference in the supports of the itemsets, common to both D_A and D_B.

Thus, much of the existing work in the database community [9, 18, 2] assume the datasets have identical schema and that access to both datasets simultaneously is possible. By utilizing the underlying structure in the datasets, we avoid making such assumptions.

Li *et al.* [16] use a variant of the mutual information between datasets D_A and D_B, modeled by sets of maximal frequent itemsets(MFIs) F_A and F_B, which is defined as $I(F_A, F_B) =$

$$\sum_{i \in F_A, j \in F_B} \frac{|i \cap j|}{|i \cup j|} log(1 + \frac{|i \cap j|}{|i \cup j|}) * min(|i|, |j|)$$

They assume an identical schema for two datasets and define the similarity between the datasets as,

$$\frac{I(F_A, F_B) * 2}{I(F_A, F_A) + I(F_B, F_B)}$$

To test for significance of similarity, they propose bootstrapping-based approaches in which disjoint pairs of subsets of the attributes are drawn at random from samples of the given datasets. The similarity between the pairs of samples are used to estimate the distribution of similarity between the two datasets. They then generalize their approach to heterogeneous datasets, of which matchings between some of the attributes of the two datasets are known. These matchings are used to identify matchings of at least ξ attributes of one dataset with those of the other.

There have been a number of graph matching algorithms, stemming from work in the field of computer vision, regarding applications like image registration, object recognition, etc. Many of the past approaches involve matching between labeled or discrete-attributed graphs [5, 6, 11, 21]. Like the solutions to many other NP-hard problems, graph matching algorithms may be *enumerative* [5, 20] or *optimization-based* [6, 21]. Most of these algorithms assume the graphs lie in the same space, which is usually low-dimensional, i.e., two or three dimensions.

The concept, "Two vertices are similar, if vertices they are related to are similar" allows recursive definition of inter-vertex similarity. This idea is used explicitly or implicitly by a number of propagation-based algorithms ([15]) for a range of applications. The recursive definition causes similarity to flow from one vertex to the other.

Blondel *et al.* [4] show that given w_A and w_B, a $|V_A| \times |V_B|$ similarity matrix S, whose real entry $s_{i,j}$ represents the similarity between vertex i of G_A and j of G_B, can be obtained as the limit of the normalized even iterates of $S_{k+1} = w_B S_k {w_A}^T + {w_B}^T S_k w_A$. Note that this model does not assume that w_A and w_B are symmetric. This algorithm has time complexity of matrix multiplication, which is currently $O(n^{2.376})$.We compare our algorithms with Blondel's algorithm.

3 Constructing a condensed model of the dataset

We represent each dataset D_A by a graph $G_A(V_A, E_A, w_A)$ where V_A is the set of subspaces found by the subspace mining algorithm and $w_A : S_A \times S_A \rightarrow \Re$ is the adjacency matrix, indicating similarity between components/subspaces in the condensed model/graph, of G_A. Depending on whether we use support or similarity of projections as the basis for comparing subspaces, we prescribe the following subspace similarity measures:

3.1 Support-based subspace similarity

Each subspace $u \in V_A$ partitions the space S_A into a clustering containing two clusters, i.e., u and $S_A \backslash u$. Accordingly, if $C_u,\ C_v$ are the clusterings yielded by subspaces $u,\ v \in V_A$, we can define $w_A(u, v)$ using $Jaccard(C_u, C_v)$ and $Rand(C_u, C_v)$ as described in Section 2.2. Additionally, we experiment with using $w(u, v) = exp(-VI(C_u, C_v))$.

3.2 Projection-based subspace similarity

Consider the case where the datasets being modeled are sets of points sampled in different proportions with respect to each other from the same mixture of multivariate distributions. Then, correctly matching these distributions using support-based subspace similarity measures, as described in Section 3.1, is unlikely. Accordingly, we seek similarity measures which use similarity of the projections of the subspaces.

We define the similarity between subspace $R \in V_A$ and a grid cell Q surrounding a point $r \in D_A$ using the Jaccard-coefficient as

$$\rho(r \in Q, R \in V_A) = \frac{1}{d_A} \sum_{i=1}^{d_A} \frac{|Q_i \cap R_i|}{|Q_i \cup R_i|} \tag{3}$$

Here, X_i refers to the set of intervals spanned by subspace X in dimension i. If dimension i is unconstrained, $|X_i| = \xi$. Using our running example from Section 1.2, $\rho(p_1 \in g_1, c_1) = \frac{1}{d_A}\left(\frac{1}{\xi} + \frac{1}{1} + \frac{1}{\xi} + \frac{1}{\xi} + \frac{1}{\xi}\right) = 0.28$
Based on the subspaces found by the subspace mining algorithm, it is possible, for example using nearest neighbors, to assign points in the dataset to subspaces. Using the assignment of points to subspaces, we have devised two similarity measures:
AVERAGE_SIMILARITY: Each subspace may be thought to be more accurately approximated by the points assigned to it. As we know the similarity between the grid cell around each point and every subspace found by the subspace mining algorithm using $\rho()$ from Equation 3, the similarity between two subspaces $u \in V_A,\ v \in E_A$ can be defined as

$$w_A(u, v) = \frac{\sum_{r \in u} \rho(r, v)}{|u|} + \frac{\sum_{r \in v} \rho(r, u)}{|v|} \tag{4}$$

From our running example in Section 1.2, $\rho(p_1 \in g_1, c_2) = 0.24,\ \rho(p_2 \in g_2, c_2) = 0.44,\ \rho(p_3 \in g_3, c_1) = \rho(p_4 \in g_4, c_1) = \rho(p_5 \in g_5, c_1) = 0.28$. Then, $w_A(c_1, c_2) = \frac{0.24+0.44}{2} + \frac{0.28+0.28+0.28}{3} = 0.62$. To ensure that $\forall u \in V_A, w_A(u, u) = 1$, we normalize by setting $w_A(u, v) = \frac{w_A(u,v)}{\sqrt{w_A(u,u) \times w_A(v,v)}}$

HISTOGRAM: Based on the coordinates of points assigned to each subspace in V_A, we estimate discrete p.d.f.s for each dimension for each subspace. If each dimension of the d_A-dimensional dataset is discretized into ξ equi-width intervals, then $u(i, j)$ corresponds to the fraction of points assigned to vertex/subspace u discretized to the jth interval in the ith dimension. Using our running example from Section 1.2, there are two points $p_1,\ p_2$ assigned to subspace c_1. Both of them are discretized to the interval 5 in the dimension d_2, i.e., [500,600). Therefore, $c_1(2, 5) = \frac{2}{2} = 1$. Accordingly,

$$w_A(u, v) = \frac{1}{d_A \xi} \sum_{i=1}^{d_A} \sum_{j=1}^{\xi} sim(u(i, j), v(i, j)) \tag{5}$$

where $sim : [0, 1] \times [0, 1] \to [0, 1]$ is a similarity function. Note that this $w_A()$ requires no normalization if we use the Gaussian or increasing weighted $sim()$ function shown below. Otherwise normalization is required. We have tested a number of symmetric similarity functions:

1. Dot product: $sim(a, b) = a \times b$
2. Gaussian weighted: $sim(a, b) = exp(\frac{-(a-b)^2}{2s^2})$
3. Increasing weighted: $sim(a, b) = \frac{1}{1+\frac{|a-b|}{s}}$

where s is a user-defined parameter controlling the spread of sim. Note the following important properties of our similarity measures:

1. They are both symmetric
2. They are independent of the number of points assigned to each subspace

4 Identifying similarities between condensed models

To find similarities between the graphs, we test four algorithms. One (**OLGA**) uses the tensor product, the next (**EigenMatch**) uses ideas from the first and Blondel's algorithm, another uses Gibbs sampling and the last uses MonteCarlo sampling (see Section 4.4).

4.1 OLGA

We combine the graphs G_A and G_B, into a single bipartite graph $G = (V_A \cup V_B, E \subseteq V_A \times V_B, \Pi)$. Π is a $|V_A| \times |V_B|$ matrix of pairwise vertex similarities.

To find Π, we construct the product graph (see function **ProductGraph** in Figure 1), $G' = (V_A \times V_B, E' \subset (V_A \times V_B) \times (V_A \times V_B), w_{A,B})$, where $w_{A,B} : E' \rightarrow \Re$ is the adjacency matrix, indicating similarity between vertices corresponding to pairs of subspaces from underlying graphs, of G'. $w_{A,B}((u,v),(u',v')) =$

$$\begin{cases} sim(w(u,u'), w(v,v')) & \text{if } sim(w(u,u'), w(v,v')) > \tau \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where τ is a user-specified threshold, used to minimize noise and limit space complexity of the algorithm. As $w(u,u'),\ w(v,v')$ depend on the specific graphs they are a part of, the weight of an edge in the product graph is high, if the weights on the corresponding edges in the underlying graphs are similar. Thus, we do not explicitly compare dimensions of vertices in the two graphs, thereby making no assumptions on identical schema. Let $S = vec(\Pi)$ (as defined in Section 1.1). Using the concept "Two vertices are similar, if vertices they are related to, are similar", then similarity between $u \in V_A$ and $v \in V_B$ is a function of all the vertices in V_A and V_B and the relationships that u and v have with them, respectively. We can write this recursively as

$$\begin{aligned} S_i(u,v) &= \sum_{u' \in V_A, v' \in V_B} w_{A,B}((u,u'),(v,v')) S_{i-1}(u,v) \\ \text{i.e., } S_i(u,v) &= w_{A,B}((u,v)) \cdot S_{i-1}(u,v) \\ \text{Then, } S_i &= w_{A,B} \cdot S_{i+1} \end{aligned}$$

where S_i denotes S at iteration i. As shown in Figure 1, we set the initial similarities, i.e., all entries in S_0 to 1.0 (line 6). We then iterate using Equation 7 (line 8). We determine convergence by checking to see if the Frobenius norm of the residual at the end of each iteration is less than a user-specified threshold ϵ (line 9).

```
ProductGraph($G, G_A, G_B$):
1.  $\forall (u,v) \in (V_A \times V_B)$ create node $(u,v)$
2.  $\forall (u,u') \in (V_A \times V_A)$
3.    $\forall (v,v') \in (V_B \times V_B)$
4.      add edge $((u,v),(u',v'))$ using Eq. 6

OLGA($G_A, G_B, \tau, k$):
5.  ProductGraph($G, G_A, G_B$)
6.  $S_0 = ones(|V_A|, |V_B|)$
7.  for i=1:k
8.    $S_i = \frac{w_{A,B} \cdot S_{i-1}}{||w_{A,B} \cdot S_{i-1}||_F}$
9.    if $||S_i - S_{i-1}||_F < \epsilon$ break
10. return Match($S_k$)

FastOLGA($G_A, G_B$):
11. Find $U_{A,1}, \lambda_{A,1}, \lambda_{A,2}$
12. Find $U_{B,1}, \lambda_{B,1}, \lambda_{B,2}$
13. if $\lambda_{A,1} \neq \lambda_{A,2}$ and $\lambda_{B,1} \neq \lambda_{B,2}$
14.   $S = U_{A,1} \otimes U_{B,1}$
15.   return Match($S$)
```

Figure 1: Matching two graphs

As we are looking for a matching between vertices from G_A to G_B, we may unstack the vector S and use the resulting $|V_A| \times |V_B|$ matrix as the adjacency matrix of the bipartite graph G, i.e., Π.

Ideally, Π is a permutation matrix which minimizes $err(f|w_A, w_B)$ (Equation 2). Typically however, Π is a real matrix. Hence, we need to *round* Π to a permutation matrix. We use the **Match** function to do the same. **Match** returns $f : V_A \to V_B$. There are a number of matching algorithms, e.g., stable matching, the Kuhn-Munkres algorithm [13], perfectionist egalitarian polygamy [15], etc. We can formulate the *rounding* as finding a matching which maximizes the sum of the weights on the edges of the matching. Finding such a matching (also called an *alignment*) is called *bipartite weighted matching*, which has earlier been optimally solved by the Hungarian algorithm [13]. This algorithm has complexity $O(\max\{|V_A|, |V_B|\})^3)$. This is equivalent to partitioning G into a number of clusters such that no cluster contains two vertices from the same graph and the total of the similarity among the vertices within each cluster is maximized. **Match**, unless otherwise mentioned, refers to the Hungarian algorithm. There are other approximate matching algorithms of lower complexity. We do not take into account the complexity of **Match** while stating complexity of the algorithms, as it is a parameter.

This idea is similar to that of Melnik *et al.* in [15]. However, they use directed, labeled graphs.

If $w_{A,B}$ is normal, it is diagonalizable. If it has a dominant eigenvalue,

$$S_i = \frac{w_{A,B} \cdot S_{i-1}}{||w_{A,B} \cdot S_{i-1}||_F} \text{ Then, } S' = \lim_{i \to \infty} S_i = \frac{w_{A,B} \cdot S'}{||w_{A,B} \cdot S'||_F} \tag{7}$$

Rearranging, $(w_{A,B} - ||w_{A,B} \cdot S'||_F \cdot I)S' = 0$, where I is the $l \times l$ identity matrix. Note, this is the characteristic equation for $w_{A,B}$. Then, $w_{A,B}$ has a dominant eigenvalue $\lambda_1 = ||w_{A,B} \cdot S'||_F$ and dominating eigenvector S'. The rate of convergence is determined by the ratio $\frac{\lambda_2}{\lambda_1}$.

If sim returns the scalar product of its inputs and $\tau = 0$, then
$w_{A,B}((u,v),(u',v')) = w(u,u')w(v,v')$ and
$w_{A,B} = w_A \otimes w_B$, as defined in Section 1.1. If $w_{A,B}$ corresponds to the tensor product, further improvements in the time and space complexity of the algorithm are possible. Accordingly, we have the algorithm **FastOLGA** in Figure 1.

It is known[8] that the set of eigenvalues of the tensor product of two matrices is the set of values in the tensor product of the eigenvalues of these matrices, i.e., $w_{A,B} = w_A \otimes w_B \Rightarrow 1 \leq i,j \leq |V_A|, |V_B|, \lambda_{w_A,i}\lambda_{w_B,j}$ is an eigenvalue of $w_{A,B}$. Hence, the dominant eigenvalue of the tensor product of $w_{A,B}$ (if it exists) is the product of the dominant eigenvalues of the w_A and w_B. This implies that convergence is achieved if both w_A and w_B have dominant eigenvalues (line 13). Similarly, the set of eigenvectors of the tensor product of two matrices is the set of values in the tensor product of the eigenvectors of these matrices. This implies that $S' = u_{A,1} \otimes u_{B,1}$. Finding a maximal matching in the tensor product of the dominant eigenvectors corresponds to projecting the longer eigenvector onto the space of the smaller eigenvector and permuting the dimensions of the former, such that their cosine similarity is maximized, i.e., aligning them.

The dominant eigenvector of an $n \times n$ matrix can be determined in $O(n^2)$ time (lines 11,12) using QR factorization [10] and the tensor product of $|V_A|$ and $|V_B|$ length vectors is computed in $|V_A||V_B|$ steps (line 14). This allows computation of S' in $O(max(|V_A|^2, |V_B|^2))$ time, i.e., faster than the Blondel algorithm.

4.2 EigenMatch

The main result of the **OLGA** algorithm is that it approximately reduces graph matching to the problem of aligning the dominant eigenvectors of the two graphs to be matched. This raises the question: why not try to align more than just the dominant eigenvectors? Accordingly, we analyze the optimization function err (Equation 2).

Note that $Tr[ww^T] = ||w||_F^2$. Then,

$$\begin{aligned}
& \min_P ||w_A - Pw_BP^T||_F^2 \\
= \quad & \min_P Tr[(w_A - Pw_BP^T)(w_A - Pw_BP^T)^T] \\
= \quad & \min_P Tr[w_Aw_A^T + w_Bw_B^T - w_APw_BP^T - Pw_BP^Tw_A] \\
= \quad & Tr[w_Aw_A^T] + Tr[w_Bw_B^T] - \min_P Tr[w_APw_BP^T + Pw_BP^Tw_A] \\
= \quad & ||w_A||_F^2 + ||w_B||_F^2 + \max_P Tr[w_APw_BP^T + Pw_BP^Tw_A]
\end{aligned}$$

As the trace of the product of two square matrices is independent of the order of multiplication, $Tr[w_A(Pw_BP^T)] = Tr[(Pw_BP^T)w_A]$. Also, $||w_A||_F^2$, $||w_B||_F^2$ are constants. Hence the problem reduces to $\max_P Tr[w_APw_BP^T]$. If w_A, w_B are normal matrices. Then, using eigendecomposition,

$$\begin{aligned}
& \max_P Tr[w_APw_BP^T] \\
= \quad & \max_P Tr[U_AD_AU_A^TPU_BD_BU_B^TP^T] \\
= \quad & \max_P Tr[(D_AU_A^TPU_BD_BU_B^TP^T)U_A] \\
= \quad & \max_P Tr[D_A(U_A^TPU_B)D_B(U_B^TP^TU_A)] \\
= \quad & \max_W Tr[D_AWD_BW^T] \text{ where } W = U_A^TPU_B
\end{aligned}$$

Blondel *et al.* use normalized even iterates of $S_{k+1} = w_AS_kw_B$ to find similarities between normal matrices w_A, w_B. We adopt this idea, so that $W_{k+1} = D_AW_kD_B$. We drop the normalization as it is a constant for a single iteration. However, instead of an iterative algorithm, we choose a good seed and utilize just one iteration, i.e., $W_1 = D_AW_0D_B$. For the seed, we use the **FastOLGA** algorithm (line 2), which aligns the dominant eigenvectors. Substituting in W, we get $D_AW_0D_B = U_A^TPU_B$. Rearranging, we get

$$P = U_AD_AW_0D_BU_B^T, \quad \text{where} \quad W_0 = \textbf{FastOLGA}(w_A, w_B) \tag{8}$$

$U_XD_X = [U_{X,1}\lambda_{X,1} \quad U_{X,2}\lambda_{X,2}\ldots]$. Thus, each eigenvector of w_A and w_B will then be weighted by its eigenvalue . Then during rounding of P, the matching algorithm will be fully cognizant of the smaller eigenvalues as well. Accordingly, we have the algorithm **EigenMatch** as shown in Figure 2.

This algorithm has the same time complexity as eigendecomposition i.e. $O(n^3)$

```
EigenMatch($G_A, G_B$):
1. $w_A = U_AD_AU_A^T,\ w_B = U_BD_BU_B^T$
2. $W_0$ =FastOLGA($w_A, w_B$)
3. $P = U_AD_AW_0D_BU_B^T$
4. return Match($P$)
```

Figure 2: Matching all eigenvectors

[10].

4.3 Mapping via Gibbs sampling

Often, while matching graphs, some nodes may ideally not have any node matched with them, e.g., occlusions in computer vision. Stochastic algorithms like Gibbs sampling, as opposed to **OLGA**, allows keeping vertices in V_A unmapped.

We use Gibbs sampling [17] to minimize the function err (Equation 2). We generate 1000 matchings (line 2), using MonteCarlo sampling as described in the Section 4.4, and estimate the background distribution of this objective function. For each such matching f, we iterate over every vertex $u \in G_A$ (line 3). We create a new mapping $f'_{u,v}$ for every vertex v in the larger graph G_B (line 4) by mapping u to v, i.e., $f'_{u,v} = f(V_A \backslash u) \cup f'(u) = v$. We define the probability of mapping u to v to be proportional to the error incurred, i.e., $p(f'_{u,v}|w_A, w_B) \propto err(f'_{u,v})$

$$\text{Then, } p(f'_{u,v}|w_A, w_B) = \frac{err(f'_{u,v}|w_A, w_B)}{\sum_{x \in V_B} err(f'_{u,x}|w_A, w_B)} \tag{9}$$

We average out these probabilities over all $f'_{u,v}$ generated in the background estimation phase, so that,

$$p(u,v) = \frac{\sum_{f'_{u,v}} p(f'_{u,v})}{\sum_{f'_{u,v}} 1}$$

Having estimated the background distribution $p()$, we generate seed mappings (line 7). For each seed, as in the background estimation stage, we estimate the probability of mapping u to v, i.e., $q(f'_{u,v})$ using RHS of Equation 9. Then we sample from $v \in V_B$, using the ratio of the seed and background distributions,

$$a(f'_{u,v}) \propto \frac{q(f'_{u,v})}{p(u,v)} \quad \text{Then, } a(f'_{u,v}) = \frac{\frac{q(f'_{u,v})}{p(u,v)}}{\sum_{\forall x \in V_B} \frac{q(f'_{u,x})}{p(u,x)}} \tag{10}$$

Thus, we try to improve the seed matching in a way that minimizes the error w.r.t. the background distribution. As we sample only one parameter at a time (the vertex u is fixed, while the vertex in V_B it is matched with is sampled), it is a Gibbs sampling algorithm.

4.4 Matching using MonteCarlo sampling

If $|V_A| >= |V_B|$, we generate a random permutation of the numbers [1,$|V_A|$] and map the first $|V_B|$ numbers of this permutation to the vertices numbered [1,$|V_B|$] of G_B. Otherwise, we swap the graphs and get the mapping in the same way. We call this *MonteCarlo sampling*.

We repeat this sampling a number of times, evaluate them using the $Zscore$ described in Section 5 and keep the one with the best $Zscore$. The number of such samples generated is controlled by the time taken to run **OLGA**. This ensures that **OLGA** and MonteCarlo sampling have the same amount of time to find the matching.

```
GibbsSampling($G_A, G_B$):
1.  for i=1:1000
2.    generate MonteCarlo matching $f$
3.    $\forall u \in V_A$
4.      $\forall v \in V_B$
5.        estimate $p(f'_{u,v}|w_A, w_B)$ using Eq. 9
6.  BestGlobalErr = $\infty$
7.  for i=1:100
8.    generate MonteCarlo matching $f$ and compute $err(f)$
9.    BestLocalErr = $\infty$
10.   for i=1:1000
11.     $\forall u \in V_A$
12.       $\forall v \in V_B$
13.         estimate $q(f'_{u,v}|w_A, w_B)$ using Eq. 9
14.       sample from $a()$ using Eq. 10 and update $f$ to $f'$
15.       BestLocalErr = min($err(f')$,BestLocalErr)
16.   BestGlobalErr = min(BestGlobalErr,BestLocalErr)
```

Figure 3: Gibbs sampling

5 Experiments

In evaluating the performance of the algorithms, we pay attention to the following measures:

1. Execution time

2. Matches: It is the number of D_B's matchable components that are correctly matched. A component in D_B is *matchable* if there exists a known, unusually similar component in D_A.

3. *Zscore*:We estimate the distribution of $err(f|w_A, w_B)$ (Equation 2) by generating a number of matchings using MonteCarlo sampling and computing the *err*. Using this distribution, the mean and variance can be determined and the scores corresponding to the mapping found by an algorithm is normalized to get the *Zscore*. Thus, the *Zscore* is the number of deviations from the mean. Very negative *Zscore*s imply the corresponding matching is very unlikely to have happened by MonteCarlo sampling (Section 4.4) and such a matching is said to have found *unusually similar* substructure.

Experiments on **OLGA**, Blondel's algorithm, MonteCarlo sampling and **GibbsSampling** were carried out on a SUN Sparc 650 MHz machine running on Solaris O/S with 256 MB RAM in C++. Blondel's algorithm, **EigenMatch, FastOLGA** and MonteCarlo sampling were also implemented on a Pentium 2 GHz machine running on Windows XP with 256MB RAM in Matlab.

5.1 Synthetic datasets

We use synthetic datasets to test the performance of our algorithms and similarity measures as dataset and algorithm parameters are varied. By generating the datasets ourselves, we can verify the correctness. Our program for generating synthetic datasets is based on that previously described in [19]. It has the following set of parameters:
1. Average number of dimensions (d)
2. Average number of points in a dataset (n)
3. Average number of embedded subspaces (k)
4. Average probability that a subspace is constrained in a dimension (c)
5. Average probability that a subspace is constrained in the same dimension as the previous subspace (o)
6. Amount of perturbation (p)
7. Type of transformation

First, $1.5k$ subspaces are generated one after the other. They are by default multivariate normal with means in each dimension $\mu(j \in [1, d])$, chosen from U[0,1000), where U[l,h) implies a uniform distribution over the interval [l,h). The standard deviation in each dimension $\sigma(j \in [1, d])$, is by default set to 20. A dimension is constrained with probability c. Two serially generated subspaces are constrained in the same dimension with probability o. Their means are constrained to be within 2 standard deviations of each other, to allow overlapping of subspaces. Unconstrained dimensions have means chosen from U[0,1000).

For $i \in \{1, 2\}$, for dataset D_i, n_i, k_i are chosen uniformly from U$(.5n, 1.5n)$ and U$(.5k, 1.5k)$ respectively. The first k_i subspaces are embedded in D_i after perturbing their parameters using a transformation. There are three types of transformations:

1. Noisy: $\forall j \in [1, d], \mu(j) = \mu(j) + U(-p, p) * 1000$
   $\sigma(j) = \sigma(j) * (1 + U(-p, p))$
2. Translation: $\mu(j) = \mu(j) + i * p * 1000$
3. Scaling: $\sigma(j) = \sigma(j) * (1 + ip/5)$

where p is the perturbation parameter. Each embedded subspace accounts for at least 1% of the total number of points. The actual number of points corresponding to a subspace is a function of the imbalance factor a, $a = \frac{\max_l \alpha_l}{\min_l \alpha_l}$ where α_l is the fraction of D_i generated using parameters of the lth subspace embedded in D_i. Noisy points, which account for 5% of the points in D_i, are multivariate uniform, i.e., each coordinate is chosen from U[0,1000).

In experiments shown below, we assume that the subspace mining algorithm finds the embedded subspaces correctly, so as to isolate the contributions of this paper. Thus, we test only the graph creation and matching algorithms described in this paper. We tested the algorithms by matching synthetic datasets having embedded subspaces. As we serially insert subspaces, for every pair of datasets, the dataset with the larger number of embedded subspaces, includes all subspaces embedded in the other dataset. The datasets have, on average $n = 1000$ points and $d = 50$ dimensions and $k = 25$ embedded subspaces, except those with $k > 40$ subspaces, which have $n = 10000$ points. Unless otherwise stated,

$c = o = 0.5,\ p = 0.03,\ a = 4.0$, we use the noisy transformation and Gaussian weighted $sim()$ function. By default, we try to map a 27-vertex graph to a 34-vertex one using **OLGA** and the HISTOGRAM similarity measure. For **OLGA**, we set $\tau = .925,\ k = 30$. We evaluate the algorithms based on the number of matches and its $Zscore$, as some parameter in the dataset is varied or the subspace similarity function is varied.

5.2 Comparison of similarity functions

We first tested the similarity functions by attempting to match each graph to itself. As expected, we found that while all the linear algebra based algorithm succeed in doing so, the MonteCarlo sampling often does not. We have not shown these results, due to space constraints.

In Figures 4,6,8 we compare **OLGA**'s performance in terms of the number of matches as some parameter, viz., p, o and c, used in generating the embedded subspaces is varied. Note that the number of matches is virtually the same for both measures, except parameter p, where HISTOGRAM performs better at $p > 0.05$. It also outperforms AVERAGE_SIMILARITY in terms of $Zscore$ as seen in Figures 5,7 and 9. This suggests that HISTOGRAM favors a more global solution as compared to AVERAGE_SIMILARITY. This occurs because it develops a profile for the entire subspace, including dimensions for which the subspace is not constrained, whereas AVERAGE_SIMILARITY takes more of a discretized approach. Also, there exist some values of these parameters, for which HISTOGRAM's $Zscore$ drops below that of the optimal matching, inspite of having a significantly lower number of matches. This happens for extreme settings of the parameters. It suggests that $Zscore$ and hence err quality measures are best suited to matching datasets having low amount of perturbation.

In Figures 10,11 we compare the effect that different transformations on the dataset have on similarity measures. We notice that AVERAGE_SIMILARITY is consistently outperformed by HISTOGRAM in the $Zscore$ category, emphasizing the robustness of the latter. In terms of the number of matches, HISTOGRAM is outperformed for the noisy transformation because again it tries to optimize globally. Thus, in general HISTOGRAM outperforms AVERAGE_SIMILARITY.

5.3 Comparison of support-based similarity functions

We discussed three functions used to compare clusterings in Section 2.2. In Section 3.1, we showed how to use them to find support-based subspace similarity. From Figure 12, Jaccard Index outperforms Rand Index and VI Metric.

5.4 Comparison of mapping algorithms

Firstly, we found experimentally that **FastOLGA** and Blondel's algorithm always arrive at the identical matching, suggesting that the similarity transformation found by Blondel's algorithm is basically the tensor product of the dominant eigenvectors. Note however that our algorithm is theoretically faster than

Blondel's. In view of this result, we show their results combined except for the timing results.

In Figures 13,14 and 15 we compare the performance of **OLGA**, **EigenMatch** and **GibbsSampling** with that of Blondel's algorithm [4] and the best matching produced in terms of $Zscore$ by MonteCarlo sampling. Note that for Figure 13 the log scale is used for the y-axis. Although **OLGA**, is the most consistent performer. The best matching produced by MonteCarlo sampling (denoted in the figures as "best MonteCarlo") performs well for matching small graphs, as it has a smaller state space to search. In Figure 14, **EigenMatch** outperforms the others in minimizing the $Zscore$ more often than not. However, **EigenMatch** is unreliable in terms of the number of matches it produces. It sometimes produces no matches, while for $k = 75$, it perfectly matches 13 vertices. This is because it attempts global optimization, in trying to align all the eigenvectors. **OLGA** by virtue of using the sim function, prunes the graph and hence tries to find unusually similar matches. Hence, it typically outperforms Blondel's algorithm. Also, note that while Blondel's algorithm converges faster than the other algorithms, it is provably slower than **FastOLGA** and produces the same results. We have verified this using our Matlab simulation, but have not shown it in the graph as they were computed on different machines and platforms.

All the algorithms have running time independent of n and d. Hence, results for these are not shown.**GibbsSampling** performs the worst of the lot in all the evaluation measures.

6 Application

With a large number of high-dimensional gene expression datasets becoming available, there is a growing need to integrate information from heterogeneous sources. For example, different clustering algorithms, designed to serve the same purpose, may be run on a dataset and we may wish to integrate output from the two algorithms. Alternatively, the same algorithm may be run on two datasets differing only slightly in experimental conditions. In the first example, the algorithm provides heterogeneity, while in the latter, it is the experimental conditions.

In our specific application, we look at two microarray datasets pertaining to the *Saccharomyces cerevisiae* (yeast) cell cycle[4]. The culture are synchronized by different mechanisms, viz., alpha factor block-release(A) and centrifugal elutriation (E). They are time series data and A has 16 samples/columns taken at 7 minute intervals, while E has 14 samples/columns taken at 30 minute intervals. Each dataset has 7680 genes/rows. The entry in the ith row and jth column of the dataset corresponds to the gene expression value for the jth time sample of gene i during the cell cycle of yeast. Microarray datasets are known to very noisy. Also, these datasets have a large number of missing values as well.

It is hypothesized that genes, which exhibit similar expression patterns may be co-regulated, i.e., having similar regulation mechanisms. Hence, we are looking for subspaces having similar expression patterns. We use SCHISM[19] to

[4]see http://www.ncbi.nlm.nih.gov/projects/geo/gds/gds_browse.cgi

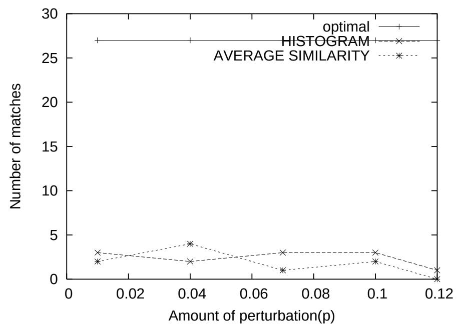

Figure 4: #(matches) v/s p

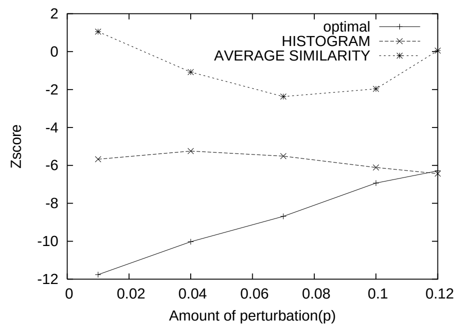

Figure 5: Zscore v/s p

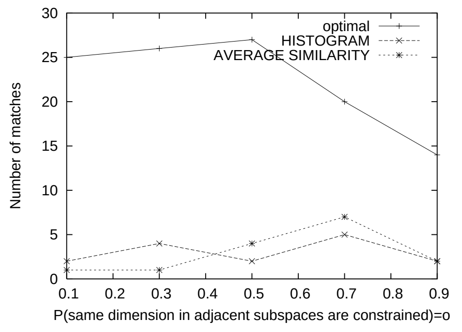

Figure 6: #(matches) v/s o

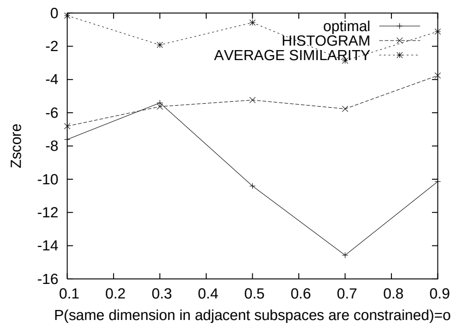

Figure 7: Zscore v/s o

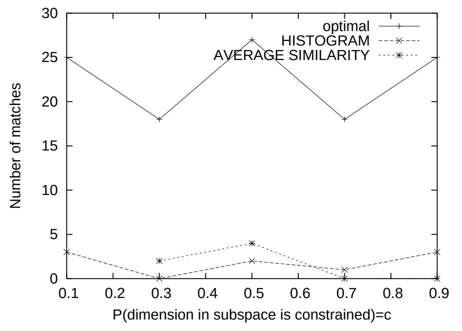

Figure 8: #(matches) v/s c

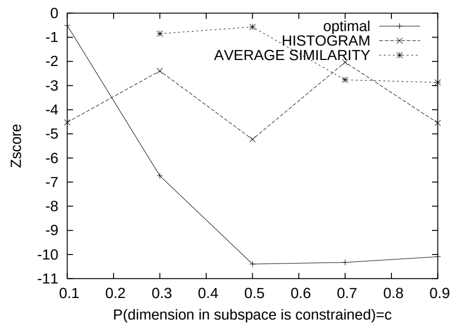

Figure 9: Zscore v/s c

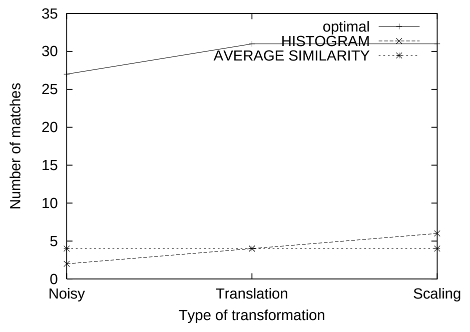

Figure 10: #(matches) v/s transformation

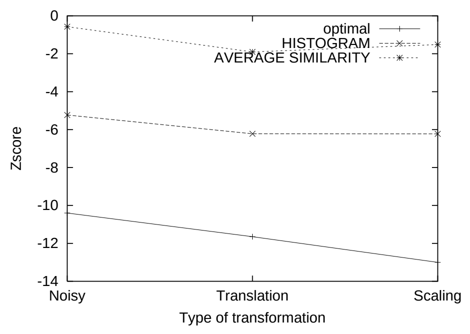

Figure 11: Zscore v/s transformation

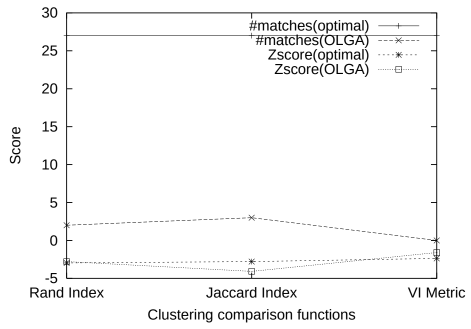

Figure 12: Clustering comparison functions

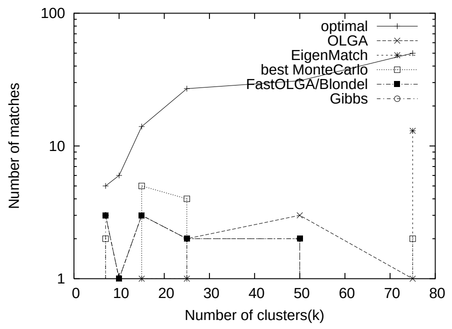

Figure 13: #(matches) v/s k

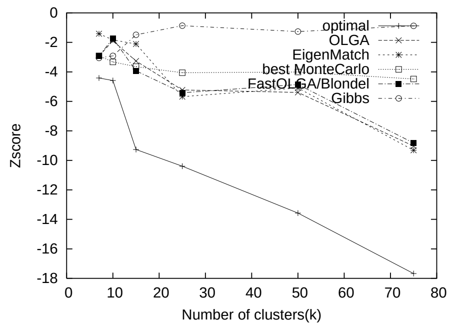

Figure 14: Zscore v/s k

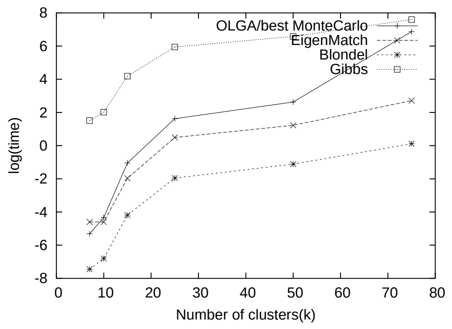

Figure 15: time v/s k

find these subspaces. We use $\xi = 3$. This discretizes gene expression values into three categories: under expressed(1st interval), normal(2nd interval) and overexpressed (3rd interval). Thus, the subspaces correspond to a subset of the genes/rows which are simultaneously either underexpressed or normal or overexpressed for some subset of the time samples/columns. SCHISM returns 87 and 94 subspaces for datasets A and E respectively. We then construct the graphs for each dataset and match the underlying subspaces/vertices using **OLGA** and HISTOGRAM. We examined the genes in the intersection of the matched subspaces to verify the efficacy of our algorithms. We submitted the list of genes in the intersection of the matched subspaces to the SGD Gene Ontology(GO) Term Finder[5] tool. This tool searches for significant shared GO terms, or parents of the GO terms, used to describe the genes in the submitted list of genes to help discover what the genes may have in common. A sample of their results are shown in Table 4.

Gene Ontology(GO) term	p-value	Genes
cellular carbohydrate metabolism	0.00072	SMI1 SUC2
carbohydrate metabolism	0.00089	
cellular macromolecule metabolism	0.04449	
ribonucleoprotein complex	0.00386	RPL32 RRP5
negative regulation of transcription	0.00567	FOB1 TFB2
negative regulation of physiological process	0.00866	
negative regulation of biological process	0.00877	
regulation of transcription, DNA-dependent	0.0315	
regulation of transcription	0.03427	
regulation of metabolism	0.01676	ACT1 SKI2
regulation of physiological process	0.01867	
regulation of biological process	0.02049	
protein modification	0.00844	HOS3 UBR1
RNA metabolism	0.01422	GLN4 YJL010C
biopolymer metabolism	0.07487	
malate metabolism	9.33E-06	MDH3 MDH2
glyoxylate metabolism	2.58E-05	
glyoxylate cycle	2.58E-05	
aldehyde metabolism	0.00033	
response to heat	0.00029	BCY1 LSP1
response to temperature	0.0004	
response to abiotic stimulus	0.03823	
response to external stimulus	0.04338	
response to stress	0.00145	TSL1 TPS1 BCY1 LSP1
response to stimulus	0.00473	
vesicle-mediated transport	0.05673	CDC48 YSC84
cellular catabolism	0.09268	

Table 4: GO-based interpretation of similar substructure

The first row of Table 4 is interpreted as follows: Genes SMI1 and SUC2 are associated with the process of cellular carbohydrate metabolism. These genes actually belong to a cluster of 7 genes but out of 7272 genes in yeast there are 196 involved in this process. Hence the p-value(measure of statistical significance)

[5]see http://db.yeastgenome.org/cgi-bin/GO/goTermFinder

is 0.00072. Further, they are also associated with carbohydrate metabolism and the p-value is 0.00089. SMI1 and SUC2 belong to a subspace of 193 genes when SCHISM is applied to A. This results in a much lower p-value and are hence not reported as statistically significant. This is true for other clusters reported too. Thus, the condensed model technique yields smaller, more statistically interesting clusters, by leveraging information from multiple sources.

A cursory examination of the GO terms associated with matchings produced by **OLGA** shows that they find genes involved in biological processes such as regulation and metabolism. The molecular functions for a number of these genes are unknown and hence there is potential for discovery.

7 Conclusions

From the $Zscore$ values obtained by the algorithms, it is obvious that the algorithms find *unusually similar* matchings with respect to MonteCarlo sampling. The p-values of the inferences from the application to the microarray datasets confirms this. It is evident that **OLGA** and **EigenMatch** succeed in finding similar subspaces based on the structure of the dataset alone, without sharing the datasets.

As part of future work, we hope to extend our algorithms to finding common substructure across multiple datasets. Also, currently our similarity measures are best suited to finding similarities between hyperrectangular subspaces. Patterns in datasets may require less restrictive descriptions, for example, coherent patterns in microarray datasets, curves, etc. We hope to develop similarity measures for such patterns as well.

References

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conf*, 1998.

[2] S. Bay, M. Pazzani. Detecting Group Differences: Mining Contrast Sets. Data Mining and Knowledge Discovery. 2001.

[3] K. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft. When is nearest neighbors meaningful? *ICDT Conf*, 1999.

[4] V. Blondel, A. Gajardo, M. Heymans, P. Senellart, P. Van Dooren. A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching. In SIAM Review, Volume 46, Number 4, pp. 647-666, 2004.

[5] H. Bunke, Error Correcting Graph Matching: On the Influence of the Underlying Cost Function. IEEE Trans. Pattern Analysis and Machine Intelligence, 21:9, pp. 917-922, Sept. 1999.

[6] M. Carcassoni, E. Hancock. Alignment using Spectral Clusters. Proceedings of the 13th British Machine Vision Conference. pages 213-222. 2002.

[7] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Math. Statistics*, 23:493–509, 1952.

[8] H. Eves, Elementary Matrix Theory, Dover publications, 1980.

[9] V. Ganti, J. Gehrke, R. Ramakrishnan, and W. Loh. A framework for measuring changes in data characteristics. In PODS, 1999.

[10] G. Golub, C. Van Loan. Matrix Computations Johns Hopkins University Press; 3rd edition, 1996.

[11] H. Kalviainen, E. Oja. Comparisons of Attributed Graph Matching Algorithms for Computer Vision. In STeP-90 Finnish Artificial Intelligence Symposium, University of Oulu. pp 354-368, 1990.

[12] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, CA, 2001.

[13] H. Kuhn. The hungarian method for the assignment problem. Naval Research Logistics Quarterly. 2:83-97, 1955

[14] M. Meila. Comparing clusterings by the variation of information. In Conference on Learning Theory, 2003.

[15] S. Melnik, H. Garcia-Molina, E. Rahm. Similarity flooding: A versatile graph-matching algorithm. In IEEE-ICDE Conference Proceedings. February 2002.

[16] T. Li, M. Ogihara, S. Zhu. Similarity testing between heterogeneous datasets. UR-CS-TR781, 2002.

[17] A. Neuwald, J. Liu, C. Lawrence. Gibbs motif sampling: Detection of bacterial outer membrane repeats. Protein Science, 4:1618-1632, Cambridge University Press, 1995.

[18] S. Parthasarathy, M.Ogihara. Clustering Homogeneous Distributed Datasets, in International Conference on Practical Applications of Knowledge Discovery and Data Mining. 2000.

[19] K. Sequeira, M. Zaki. SCHISM: A New Approach to Interesting Subspace Mining. In IEEE ICDM Conference Proceedings, 2004.

[20] L. Shapiro, M. Haralick. A Metric for Comparing Relational Descriptions. IEEE Trans. Pattern Analysis and Machine Intelligence, 7:1, pp. 90-94, Jan. 1985.

[21] B. Van Wyk, M. Van Wyk. Orthonormal Kronecker Product Graph Matching. Lecture Notes In Computer Science. Vol 2726, pp 107-117, 2003.

[22] S. White, P. Smyth. Algorithms for estimating relative importance in networks. In Proceedings of the ACM SIGKDD Conference, pages 266-275, 2003.