EXPLORING SIMILARITIES IN HIGH-DIMENSIONAL DATASETS

By

Karlton Sequeira

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Major Subject: Computer Science

Approved by the
Examining Committee:

Mohammed Zaki, Thesis Adviser

Kristin Bennett, Member

Petros Drineas, Member

Daniel Freedman, Member

Rensselaer Polytechnic Institute
Troy, New York

August 2005
(For Graduation August 2005)
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 3183647
Copyright 2005 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.
To my parents
CONTENTS

LIST OF TABLES ......................................................... vi
LIST OF FIGURES ..................................................... vii
ACKNOWLEDGMENT ..................................................... ix
ABSTRACT .............................................................. x

1. INTRODUCTION .................................................... 1
   1.1 Motivating applications of finding similarities across datasets ...................................... 2
   1.2 Notation .......................................................... 3
   1.3 Challenges posed by identification of similarities across datasets ................................. 5
   1.4 Proposed high-level solution and contributions ............................................................. 7

2. HISTORICAL REVIEW ............................................. 9
   2.1 Generating a condensed representation of the dataset .................................................... 9
      2.1.1 Finding the components of the condensed representation ........................................... 10
      2.1.1.1 Density-based clustering ................................................................. 11
      2.1.1.2 Projected Clustering ................................................................. 14
      2.1.1.3 Biclustering ................................................................. 15
      2.1.2 Finding relationships across the components of the model ...................................... 15
   2.2 Finding similarities across condensed representations ............................................... 19
      2.2.1 Dataset similarity measures ................................................................. 19
      2.2.2 Graph matching .............................................................................. 20
      2.2.2.1 Enumerative/Search-based graph matching algorithms ....................................... 21
      2.2.2.2 Optimization-based graph matching algorithms ........................................... 23
      2.2.3 Flow-based algorithms ................................................................. 24
   2.3 Finding similarities between multiple datasets ............................................................. 26

3. CONDENSED REPRESENTATIONS OF DATASETS ............... 28
   3.1 Finding "interesting" subspaces in the dataset ............................................................ 29
      3.1.1 Interestingness Measure ................................................................. 30
      3.1.2 Chernoff-Hoeffding bound ................................................................. 32
      3.1.3 The SCHISM Algorithm ................................................................. 35
3.2 Representing the dataset as a graph .................. 43
  3.2.1 Support-based subspace similarity ................. 43
  3.2.2 Projection-based subspace similarity ............... 43
    3.2.2.1 AVGSIM .................................. 44
    3.2.2.2 HIST .................................. 44

4. FINDING SIMILARITIES ACROSS CONDENSED REPRESENTATIONS OF PAIRS OF DATASETS ...................... 46
  4.1 MonteCarlo Match .................................. 47
  4.2 OLGA ............................................. 47
  4.3 Mapping via Gibbs sampling .......................... 51
  4.4 RandomMatch ...................................... 52

5. EXPERIMENTS & RESULTS .................................. 57
  5.1 Performance measures ................................. 57
    5.1.1 Performance measures for condensed model representation .. 57
    5.1.2 Performance measures for finding similarities across pairs of condensed models ............... 58
  5.2 Datasets .......................................... 60
    5.2.1 Synthetic datasets .................................. 60
    5.2.2 Real datasets .................................. 62
  5.3 Results of condensed representations of datasets .................. 64
    5.3.1 On synthetic datasets ............................ 65
      5.3.1.1 Effect of varying dataset parameters ........... 65
      5.3.1.2 Effects of varying algorithm parameters .......... 70
    5.3.2 On real datasets ................................ 73
  5.4 Results of finding similarities across condensed models of pairs of datasets ..................... 79
    5.4.1 On synthetic datasets ............................. 79
      5.4.1.1 Interaction between type of transformation and similarity measure .................. 79
      5.4.1.2 Effect of varying dataset parameters on #(matches) .................................. 81
      5.4.1.3 Effect of varying dataset parameters on Zscore .................................. 81
      5.4.1.4 Effect of varying dataset parameters on #(matches) .................................. 83
      5.4.1.5 Scalability of algorithms ........................ 84
    5.4.2 On real datasets .................................. 86
6. CONCLUSIONS AND FUTURE WORK ........................................ 92
    6.1 Conclusions .......................................................... 92
    6.2 Future Work .......................................................... 92
        6.2.1 Extending the solution to multiple graphs .......... 93
            6.2.1.1 Extending FastOLGA .............................. 93
            6.2.1.2 Extending the solution to multiple graphs .... 94

BIBLIOGRAPHY ............................................................... 97
# LIST OF TABLES

3.1 Example DB: Real & Discretized ........................................... 36
3.2 Seed Subspaces: Real & Discretized ................................... 37
5.1 Confusion Matrix for PenDigits Data Set .............................. 74
5.2 Performance of matching algorithms on breast cancer datasets ... 74
5.3 Statistically significant subspaces from 2003-04 NBA season .... 76
5.4 Performance of SCHISM and CLIQUE on breast cancer datasets .. 77
5.5 GO-based interpretation of genes clustered by SCHISM ............ 78
5.6 Structurally similar clusters from two NBA seasons ............... 88
5.7 Structurally similar clusters from schematically different breast cancer datasets ....................................................... 88
5.8 GO-based interpretation of similar substructure .................... 90
LIST OF FIGURES

1.1 Schematic of high-level solution ........................................ 7
2.1 An example for which dimension reduction via PCA fails .......... 11
3.1 The original sets of points in their own domains ..................... 28
3.2 Find interesting subspaces in the two datasets and assign points .... 29
3.3 Construct graphs for each dataset using the found subspaces ........ 29
3.4 Effect of the number of constrained dimensions on $thresh$ .......... 31
3.5 The SCHISM Algorithm ................................................. 35
3.6 Mining Interesting Subspaces .......................................... 39
3.7 Lattice of running example ............................................... 41
3.8 Assigning Points in SCHISM ............................................ 42
4.1 Input: graphs for each dataset using the found subspaces .......... 46
4.2 Output: Matched vertices from the two graphs ....................... 47
4.3 OLGA, FastOLGA: Matching two graphs ............................ 49
4.4 Gibbs sampling .................................................................. 53
4.5 RandomMatch: Matching two subgraphs .............................. 54
4.6 PruneSeed, ExpandSeed: Refining the match ......................... 56
5.1 Histogram of Zscores for random matchings between 8-vertex and 21-vertex graphs .................................................. 59
5.2 Set parameters for subspaces to be embedded ....................... 61
5.3 Generate datasets by embedding subspaces .......................... 63
5.4 Effect of dataset size ($n$) on SCHISM's performance ............. 65
5.5 Effect of data dimensionality ($d$) on SCHISM's performance .... 66
5.6 Effect of subspace size imbalance ($\kappa$) on SCHISM's performance .... 67
5.7 Effect of subspace dimensionality ($c$) on SCHISM's performance .... 67
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.8</td>
<td>Effect of fraction of overlapping constrained dimensions (o) on SCHISM's performance</td>
</tr>
<tr>
<td>5.9</td>
<td>Effect of constraining means in adjacent embedded subspaces to within 2σ</td>
</tr>
<tr>
<td>5.10</td>
<td>Effect of number of embedded subspaces (k) on SCHISM's running time for different background distributions</td>
</tr>
<tr>
<td>5.11</td>
<td>Effect of number of embedded subspaces (k) on entropy of SCHISM for different background distributions</td>
</tr>
<tr>
<td>5.12</td>
<td>Effect of hyperrectangle subspace density (w) on SCHISM's performance</td>
</tr>
<tr>
<td>5.13</td>
<td>Effect of Gaussian subspace density (w) on SCHISM's performance</td>
</tr>
<tr>
<td>5.14</td>
<td>Effect of interestingness threshold (τ) on SCHISM's performance</td>
</tr>
<tr>
<td>5.15</td>
<td>Effect of ξ on SCHISM's performance</td>
</tr>
<tr>
<td>5.16</td>
<td>Effect of thresh on performance: CLIQUE vs. SCHISM</td>
</tr>
<tr>
<td>5.17</td>
<td>Ratio of Zscores v/s type of dataset transformation</td>
</tr>
<tr>
<td>5.18</td>
<td>#(matches) v/s type of dataset transformation</td>
</tr>
<tr>
<td>5.19</td>
<td>Zscore v/s c, subspace dimensionality parameter</td>
</tr>
<tr>
<td>5.20</td>
<td>Zscore v/s o, overlap parameter</td>
</tr>
<tr>
<td>5.21</td>
<td>Zscore v/s p, amount of perturbation</td>
</tr>
<tr>
<td>5.22</td>
<td>#(matches) v/s c, subspace dimensionality parameter</td>
</tr>
<tr>
<td>5.23</td>
<td>#(matches) v/s o, overlap parameter</td>
</tr>
<tr>
<td>5.24</td>
<td>#(matches) v/s p, amount of perturbation</td>
</tr>
<tr>
<td>5.25</td>
<td>Zscore v/s k, number of embedded subspaces</td>
</tr>
<tr>
<td>5.26</td>
<td>#(matches) v/s k, number of embedded subspaces</td>
</tr>
<tr>
<td>5.27</td>
<td>log(time) v/s k, number of embedded subspaces</td>
</tr>
<tr>
<td>6.1</td>
<td>GVY algorithm for minimum multicut</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENT

A Ph.D. thesis is probably the hardest thing I’ve done in my life. And to stay motivated through it for five years has been far easier thanks to the encouragement and support of others.

I would hence like to thank my adviser Prof. Mohammed Zaki for the encouragement and ideas he provided during this thesis. He provided me with a large amount of latitude in choosing my research topic and I am grateful to him for that. Also, I must credit him with nurturing my interest in data mining enough, to choose this field.

I would also like to thank a number of students at RPI, who provided me with inspiration and a whole lot of patience, especially Sujit Biswas, Gautam Kunapuli and Arvind Venkatesan.

I also am grateful to Prof. Petros Drineas for some stimulating discussions.

Finally, I would like to thank my family, especially my parents, who made me all I am today, my brother and my sisters for their advice and my dear nieces, who never fail to inspire me with their curiosity.
ABSTRACT

Very often, 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 representation of their datasets. If some subset of the condensed representations of such datasets, from different sources, may be found to be unusually similar, policies successfully applied to one may be considered for application to the others.

In this dissertation, we tackle the problem of finding similarities across high-dimensional datasets. We propose a framework, wherein we use condensed representations of the datasets to obfuscate details and limit noise. We provide algorithms to find interesting regions within datasets which become components of the condensed representations. We propose similarity measures for these components. We then use a graph-matching based formulation to find structurally similar components across the condensed representations of the datasets. As opposed to some earlier algorithms, we show that it is possible to match a subgraph in one graph to a subgraph in another.

We test our algorithms on a wide array of synthetic and real datasets. We make a number of discoveries. We find that structure-based similarity enhances amplification of weaker patterns. It allows discovery of patterns via integration of datasets having possibly differing schema.
CHAPTER 1
INTRODUCTION

Data mining is the process of automatic discovery of novel, useful and understandable patterns and statistically significant structures from large amounts of data. It is an interdisciplinary field merging ideas from statistics, machine learning, database systems and data-warehousing, high-performance computing, as well as visualization and human-computer interaction. It has been engendered by the phenomenal growth of data in all spheres of human endeavor, and the economic and scientific need to extract useful information from the collected data.

While much effort has been spent in discovering patterns like anomalies, itemsets, association rules, sequences and trees [40], the intuitively harder problems like similar substructure discovery, are only recently, being aggressively tackled.

Often, data may be collected by a number of sources. These sources may be geographically far apart. If the entire dataset is to be communicated to a central processing region, considerable time may be required. It may be preferable instead, to share condensed representations of the datasets. Similarly, for reasons like confidentiality agreements, local sources of noise and variation, etc., it may be required to use condensed representations of datasets, which obfuscate individual details, while conveying structural information about the datasets. If unusually similar substructure can be detected from the condensed representations of some of the datasets, then this substructure may be exploited in many ways. In this chapter, we present a few applications which would benefit from the exploration of similarities across datasets, as motivation to solve the problem. We then highlight some of the challenges posed by this problem and finally discuss the basic steps we have split the problem into.
1.1 Motivating applications of finding similarities across datasets

Two hospitals may collect information about their colon cancer patients, which they cannot share with each other to avoid breach of patient confidentiality laws. However, they may be able to exchange statistical information about their patients with each other, without violating the confidentiality agreements. If a subset of their patients being treated for colon cancer are found to be unusually similar, then successful treatments employed by one hospital for that subset of colon cancer patients, may be considered by the other hospital.

Similarly, 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. This is analogous to the problem of finding contrast sets [10]. Note that determining similarities and dissimilarities across snapshots of a dataset taken over multiple time intervals can help in identifying how the dataset characteristics evolve over time [33].

The development of high-throughput microarray technology permits the recording of expression levels of thousands of genes under hundreds of experimental conditions simultaneously. One of the applications of microarray data, is the identification of the subsets of genes in the genome of an organism, that are involved in the various biological processes of that organism [35, 71, 80]. A typical solution to this problem is to cluster the genes of that organism, using their recorded expression levels across a set of experimental conditions. One technique for overcoming the inherent noise and variation in this domain, is the use of replication of experiments, i.e., if a similar subset of genes, cluster across a set of experiments carried out on different microarray chips or in experiments conducted by different researchers, it suggests that the cluster is not merely an effect of noise. There exist a number of publicly available microarray datasets for a number of genomes, collected by biologists from a number of laboratories. If clustering such datasets, corresponding
to a single genome, but originating from different sources, yields a cluster common across all the datasets, then this cluster is less likely to be a noise-influenced, biologically uninteresting cluster. As these datasets have already been collected by a diverse set of researchers, they may have different schema, different sources of noise, etc. Similarly for time-series experiments, datasets may be acquired for the same genes/probes using different synchronization methods or different number of sampling points or different rates of sampling.

If the datasets are represented by graphs, then the problem is related to the graph similarity problem which is a well studied one. This problem has many applications such as face recognition [24], tumor detection [11], schema matching [59], etc. Thus, there are numerous applications, which will benefit from frameworks designed to identify similarities across datasets.

1.2 Notation

Consider dataset $D_A$ having $d_A$ dimensions. If $S_{A,i}$ is the domain of the $i$th 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 $D_A = \{x_i : i \in [1,|D_A|], x_i \in S_A\}$. Similarly, $D_B = \{y_i : i \in [1,|D_B|], y_i \in S_B\}$. If the domain $S_{A,i}$ of each dimension is divided into $\xi$ equi-width intervals, then $S_A$ has a grid superimposed over it. Accordingly, we define a subspace\(^1\) as a grid-aligned\(^2\) hyperrectangle $[l_1, h_1] \times [l_2, h_2] \times \ldots \times [l_d, h_d]$. $\forall i \in [1,d], [l_i, h_i] \subseteq S_{A,i}$. $l_i = (a_iS_{A,i})/\xi$, $h_i = (b_iS_{A,i})/\xi$, where $a_i, b_i$ are non-negative integers, $a_i < b_i \leq \xi$. If $[l_i, h_i] \subseteq S_{A,i}$, the subspace is said to be constrained in dimension $i$, i.e., the subspace does not span the entire domain of the dimension $i$. A $p$-subspace is a subspace, constrained in $p$ dimensions and is denoted as $S_p$. A subspace, which is constrained in all the dimensions to a single interval, i.e., $\forall i \in [1,d]$, $b_i - a_i = 1$, is referred to as a grid cell.

If the dataset $D_A$ can be decomposed into $|V_A|$ subspaces, the inter-relationships between these subspaces can be expressed by a $|V_A| \times |V_A|$ matrix, $w_A : S_A \times S_A \rightarrow \mathbb{R}$.

We use the following matrix notation in this thesis. Let $A$ and $B$ be two matrix-

---

\(^1\)The word ‘subspace’ has a somewhat different meaning from the linear algebra literature. We provide the meaning commonly used in data mining literature

\(^2\)grid-aligned subspaces are essential for interpretability
ces defined as: \( A = (a_{ij})_{1 \leq i,j \leq m,n}, \ B = (b_{kl})_{1 \leq k,l \leq p,q} \). If \( m = n \), \( \text{Tr}[A] = \sum_{i \in [1,m]} a_{i,i} \) is the trace of \( A \). \( A^T \) refers to the transpose of \( A \). \( ||A||_F = (\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2)^{1/2} \) is called the Frobenius norm of \( A \). \( \text{ones}(m,n) \) returns a \( m \times n \) matrix containing all ones. \( \text{zeros}(m,n) \) returns a \( m \times n \) matrix containing all zeros. The tensor product\(^3\) of \( A \) and \( B \) is a \( mp \times nq \) matrix, and is defined as

\[
A \otimes B = \begin{pmatrix}
a_{1,1}B & a_{1,2}B & \cdots & a_{1,n}B \\
a_{2,1}B & a_{2,2}B & \cdots & a_{2,n}B \\
a_{m,1}B & a_{m,2}B & \cdots & a_{m,n}B
\end{pmatrix}
\]

Unless otherwise specified, the notation \( |x| \) denotes either the magnitude of a complex number, if \( x \) is a complex number. Or it denotes the length of vector \( x \), if \( x \) is a vector. Or it denotes the cardinality of set \( x \), if \( x \) is a set. A \( n \times n \) matrix \( X \) is called normal, if it can be written as, \( X = U_X D_X U_X^T \). \( U_X \) 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 \( i \)th eigenvalue, \( |\lambda_{X,i}| \) denotes its magnitude, 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 \( S = [s_1 \ s_2 \ \ldots] \), where \( s_1, s_2, \ldots \) are column vectors, \( \text{vec}(S) \) creates a column vector by stacking its column vectors one below the other, so that \( \text{vec}(S) = [s_1^T \ s_2^T \ \ldots]^T \).

Let \( P \) be the function, which takes as argument a mapping, \( f : V_A \rightarrow V_B \), between two sets of vertices/subspaces of datasets \( D_A \), \( D_B \), and returns a permutation submatrix\(^4\), i.e., a \( |V_A| \times |V_B| \) matrix, such that

\[
P_f(u \in V_A, v \in V_B) = \begin{cases} 
1 \quad \text{if} \ f(u) = v \\
0 \quad \text{otherwise}
\end{cases} \quad (1.1)
\]

If \( f \) is a one-to-one mapping and if \( |V_A| \leq |V_B| \), the rows (columns) of \( P \) are orthogonal to each other and \( PP^T = I(P^TP = I) \).

While finding the similarities across the datasets, we want to find a mapping

---

\(^3\)The tensor product is also called the matrix direct product or Kronecker product

\(^4\)Typically, a permutation matrix is a square matrix
$f$, such that it minimizes the error of the mapping, $\text{err}(f)$, which we define as

$$\text{err}(f|w_A, w_B) = ||w_A - P_f w_B P_f^T||_F$$  \hspace{1cm} (1.2)$$

where $w_A, w_B$ denote the weighted adjacency matrices of the graphs corresponding to the datasets $D_A$ and $D_B$, respectively.

A mapping $f$, from a set of subspaces corresponding to $w_A$, to a subset of subspaces, corresponding to $w_B$ is unusually similar, if the probability of finding another mapping $f'$ between these subsets, by MonteCarlo sampling as described in Sec. 4.1, such that $\text{err}(f'|w_A, w_B) < \text{err}(f|w_A, w_B)$, is very low.

### 1.3 Challenges posed by identification of similarities across datasets

A dataset may be considered to 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. For example, microarray data collection involves many steps, e.g., reverse transcription, mRNA extraction, scanning, image analysis, etc., each of which, is a source of noise and variation. Also, there exists considerable variation across identical experiments on different cultures of the same organism. Such noise and variation may cause a random subset of genes to cluster well, although this subset may be biologically uninteresting. There may be missing values as well. Further, unlike typical clustering problems, each gene may participate in a number of biological processes [9] motivating soft clustering. We currently assume that the dataset spaces $S_A$ and $S_B$, are mixtures of nominal and continuous variables. The datasets may be subject to translational, rotational and scaling effects as well. For example, the datasets may correspond to pictures of a car moving perpendicular to the line of vision of a still camera. In such a case, the distributions corresponding to objects which should map to each other in the two pictures will have some difference in their means, i.e., a translational effect. Similarly, 3-d coordinates for related proteins need not
be measured with their axes optimally aligned and hence there is a rotation effect. Finally, if instead, the car moves towards the camera, then two pictures taken at different distances from the car, have a scaling effect. Note that in each of these effects, the structure is preserved. This motivates the need for models which use structural comparison of the datasets.

The problem of finding similarities across multiple datasets is especially challenging due to a number of reasons:

1. Transformations: The datasets to be compared may be subject to linear transformations, i.e., similarity transformations like orthogonal rotation, translation, permutation, reflection, dilation or contraction and general linear transformations like shearing and stretching. In domains like computer vision, due to slight transformations like rotations, some nearer opaque objects may obstruct vision of more distant objects. These are called “occlusions”. Matching such images taken from different angles poses a problem as some points in one image may have no corresponding point in the other image.

2. Dimensionality: High dimensional datasets are inherently sparse and traditional distance metrics which treat every dimension with equal importance have little meaning, as 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 [13]. When, considering the distance in “full-dimensional” space, this lack of contrast induces meaningless results from algorithms using such similarity measures as a building block for application to high-dimensional datasets. Clustering algorithms using such distance metrics, e.g., K-means, hierarchical clustering, etc., are called “full-dimensional” clustering algorithms.

3. Complexity: The datasets may be generated from mixtures of distributions which may be neither identical nor completely distinct, i.e., partially overlapping. This makes the problem reducible to that of inexact graph matching, which is known to be NP-complete [1]. This means that some distributions
in one dataset will have no corresponding distribution in the other, in which they may be occluded.

4. Heterogeneity: The datasets may have different schema, i.e., different dimensional spaces and dimensionality and there is no automatically known correspondence between the two sets of dimensions. Hence, similarity measures implicitly dependent on the identity of the dimensions are naturally precluded.

1.4 Proposed high-level solution and contributions

![Diagram of high-level solution]

**Figure 1.1: Schematic of high-level solution**

Fig. 1.1 illustrates our approach to detection of similarities across datasets. Our input is a set of datasets $D = \{D_i : i \in [1,k]\}$. To identify similarities across the datasets, we identify two high-level subproblems, viz.,

- **Generating a condensed representation of the dataset:** This affords a number of advantages:
  1. It allows sharing of statistical information about the datasets.
  2. It helps to filter out noise and outliers
  3. It allows for normalization and dataset scaling.
The condensed representation is based on the detection of “interesting” subspaces in the dataset. The representation is built by computing a weighted similarity graph $G_i$ between subspaces for each dataset. These are typically graphs as shown in Fig. 1.1, by edges with dashed lines.

- **Finding similarities across the condensed representations of the datasets:** Merely finding a function $\textit{Sim} : S_A \times S_B \to \mathbb{R}$ gives us an overall impression of how similar the datasets are. It fails however to convey what parts of the datasets are similar. Two datasets may be very similar locally, but very dissimilar in the global sense, i.e., they might have as much global similarity as two perfectly random datasets. In Fig. 1.1, structurally similar vertices, across the condensed representations of the datasets, are shown connected by bidirectional solid arrows.

We aim to counter the challenges mentioned in Sec. 1.3 as follows:

1. **Complexity:** We counter the problem of complexity by using condensed representations of the datasets, i.e., graphs, to limit the size of the input. In Chapter 3, we describe an algorithm called **SCHISM** to mine the components of condensed representations. In Chapter 4, we propose algorithms using randomization, viz. **RandomMatch**, to limit the solution space to a few good seeds, which we then refine.

2. **Dimensionality:** **SCHISM** searches for ‘interesting’ subspaces, i.e., subspaces which show strong concentration vis-a-vis the high entropy, multivariate uniform distribution. Such a definition of ‘interesting’ precludes search in high-entropy dimensions. In Chapter 3, we propose measures of similarity between components found by **SCHISM**.

3. **Transformations and heterogeneity:** By defining similarity between components of condensed representations of datasets as a function of the structural position of the components in the condensed representations, many of the effects of transformations can be nullified. Also, if the datasets are closely related, the effects of heterogeneity in schema can be countered too, as shown by Melnik et al. [59].
CHAPTER 2
HISTORICAL REVIEW

In this chapter, we discuss the large body of work related to our sub-problems, viz. generating a condensed representation of the dataset and finding similarities across such representations. The literature stems from a range of fields such as information theory, statistics, data mining, graph matching and fix-point computation-based algorithms. While the first two sections deal with the literature pertaining to the problem of finding similarities across two datasets, the last section reviews literature which seeks to identify similarities in multiple datasets.

2.1 Generating a condensed representation of the dataset

To generate a condensed representation of a dataset, we need to model it. A dataset may be modeled by supervised learning models or unsupervised learning models. Supervised learning models (regression models, classification models, etc.) use labeled data, i.e., each point in the dataset has a discrete label associated with it. Hence these models are not suitable for our problem as the points in our dataset are unlabeled. In unsupervised learning models (e.g., density estimation, clustering, maximal frequent itemsets [40]), the inter-relationships between the components of the model need to be made explicit. Accordingly, the task of generating condensed representations for a dataset is divided into two sub-problems, viz.

1. Finding the components of the model, i.e., the condensed representation

2. Finding intra model similarities, i.e., relationships between the components of the condensed representation of a single dataset.

As the datasets must be compared automatically, an alternate, more powerful discrete data structure than the two dimensional array is sought to represent the dataset. Graphs are known to be extremely effective in such a function. Therefore, we treat a component in the dataset as a vertex in a graph and we propose some similarity measures to weight the edges between these vertices.
2.1.1 Finding the components of the condensed representation

One of the proposed solutions to the problem of high-dimensional clustering, as described in Sec. 1.3, is designing new distance metrics [6]. Another is reducing the dimensionality [28] and then running a traditional “full-dimensional" clustering algorithm, like k-means or hierarchical clustering [40], on the lower-dimensional dataset. Dimension reduction techniques are of two types: feature selection, in which we aim to find linear or non-linear combinations of the original set of dimensions and variable selection, in which a subset of the dimensions from the original set are selected. Prominent among the feature selection methods are the Karhunen-Loève transformation (KLT) or principal component analysis (PCA) or singular value decomposition (SVD), which project the dataset from the original \( d \) to a \( k \) dimensional space, where \( k \ll d \), and each new dimension is a linear combination of the original dimensions; after this clustering is done using only the \( k \) dimensions. Such a strategy may be inappropriate since the transformed feature space of the resultant clusters may be hard to interpret. Also, data is only clustered in a single \( k \)-dimensional space.

In addition, the assumption that the variance of data is maximized in mutually orthogonal directions, which are linear combinations of the underlying dimensions, may be inapplicable to a number of datasets. In CLIQUE [8] and DOC [68], examples are cited, in which KLT does not reduce the dimensionality without trading off considerable information, as the dataset contains subsets of points which lie in different and sometimes overlapping lower dimensional subspaces. For example, in Fig. 2.1, the data points (black dots) are distributed uniformly about the X-axis but are constrained to certain intervals of the Y-axis. The corresponding eigenvectors are denoted by dashed arrows \( (U_1 \text{ and } U_2) \), where \( U_1 \) is the dominant eigenvector. The second eigenvector is not so small in comparison with the dominant one. Hence, dimension reduction via PCA/KLT may not be so effective for such datasets. On the other hand, the thin rectangular boxes surrounding the points, perpendicular to the Y-axis, correspond to subspaces, as per our definition in Chapter 1 and these capture the variation far better than the axes of PCA. Also, these subspaces are constrained to a single dimension, i.e., Y, thereby realizing dimension reduction.

Another method of dimension reduction is feature selection, in which some of
Figure 2.1: An example for which dimension reduction via PCA fails

the dimensions are selected heuristically without transformation [17]. This removes the problem of interpretability, but still only a fixed subspace is used for clustering. There also exists the method of multidimensional scaling [52], which uses a similarity matrix to infer the underlying dimensionality. Here too, the transformed feature space requires expert interpretation. These challenges have caused the focus of much recent work in clustering to shift towards finding the interesting subspaces within a high dimensional space [4, 5, 8, 21, 22, 62, 48, 68]. Other challenges encountered in subspace mining are that subspaces may share dimensions as in the example in Fig. 2.1. Furthermore, they may also share objects, i.e., they may overlap. For example, consider the checkerboard pattern.

2.1.1.1 Density-based clustering

Prior to delving into the details of the density-based approaches, we state a few relevant definitions:

Definition 1. Subspace support: It is the fraction of points in the dataset, that lie in a subspace.

There exist multiple definitions for “dense” subspaces. Given the definition of a “dense” subspace, we may define a maximal dense subspace as follows:

Definition 2. A maximal dense subspace is a dense subspace, which encloses no
dense subspace, i.e., if $S$ is a dense subspace, there exists no dense subspace $S'$, such that $S' \subset S$.

Agrawal et al. [8], proposed CLIQUE, which discretized the domain of each of the $d_A$ dimensions of dataset $D_A$, into a user-specified number, $\xi$, of equal-width intervals. As each dimension is discretized, the corresponding intervals are analogous to items and each point is analogous to a transaction from the market basket terminology [40]. The coordinates of each point after discretization are analogous to the presence of the corresponding items in that transaction. Thus, a $p$-subspace is analogous to a $p$-itemset and hence, finding interesting subspaces or clusters is analogous to mining MFIs. CLIQUE uses subspace support, as earlier defined, to denote the density of a subspace. Those subspaces having support above a minimum support threshold (a user-specified constant $s$) are retained and are said to be ‘dense’. Dense subspaces are considered to be “interesting”. Using this definition of dense, if any $p$-subspace is dense, then all $(p + 1)$-subspaces enclosed by it, are also dense. This is the Apriori principle [7], used in CLIQUE to prune the search for dense subspaces. Initially, all 1-subspaces are considered as candidate dense 1-subspaces. There are then a series of iterations to find maximal dense subspaces. In the $k$th iteration, the dataset is scanned to find all dense $k$-subspaces from the candidate $k$-subspaces. Dense $k$-subspaces which are identically constrained in $k - 1$ dimensions, have as their intersection, a $(k + 1)$-subspace, which is tested for denseness in the $(k + 1)$th iteration. Thus, a bottom-up breadth first search (BFS) for maximal dense subspaces is carried out. To prune their search at a faster rate, they use the minimum-description length (MDL) principle as a heuristic, thereby making it an approximate search. They then merge dense subspaces sharing common faces, and use covering algorithms to mine the minimal descriptions of the subspaces.

Instead of support, Cheng [22], proposed using entropy as a measure of subspace interestingness. They mine subspaces satisfying an entropy threshold $\omega$. Nagesh et al. in MAFIA [62], partition each dimension into a large number of equi-width intervals. If the difference between the modes corresponding to the histograms for adjacent intervals are less than $\beta$ apart, they are merged, yielding intervals of variable width depending on the distribution of points. An interval is
considered ‘dense’, if the number of points in it exceeds the threshold \((aan)/|S_{A,i}|\), where \(n\) is the number of points in the dataset and \(\alpha\) is a user-specified parameter, called the cluster dominance factor and \(|S_{A,i}|\) is the range of the attribute of the \(i\)th dimension of dataset \(A\). Here, \((aan)/|S_{A,i}|\) corresponds to the number of points expected to lie inside the interval of width \(a\) in the \(i\)-th dimension, which has range \(|S_{A,i}|\). Using adaptive width intervals minimizes rigidity of grid-based clustering employed by CLIQUE.

Kailing et al. [47] suggest using a sample of the points in the dataset. They generate dense subspaces enclosing each point of the sample if it is a core object, i.e., if it has more than \(MinPts\), a user-specified threshold, points within a threshold radius \(\epsilon\). The subspaces are then assigned a quality rating, which takes into account the number of dimensions in which the subspace is constrained and this rating is used to prune lower quality subspaces. By providing a rating, it is only possible for the user to determine the relative interestingness of a subspace w.r.t. another subspace. It is not easy for the user to know the absolute interestingness of the subspace, e.g., if a dataset has nearly uniform distribution in all dimensions, it is unlikely that the most interesting subspace is interesting, in an absolute sense. In DBSCAN [31], a point \(p\), is density reachable from a core object \(q\), if there exists a chain of core objects from \(p\) to \(q\), such that each core object is within the \(\epsilon\)-neighborhood of the next in the chain, and connected to \(q\), if point \(o\) exists, such that \(p\) and \(q\) are reachable from \(o\). A cluster then, has points pairwise connected and excludes no points reachable from those in it. In later work [15], Bohm et al. extend DBSCAN to mine subspace clusters by generalizing the concepts of density reachability and connectivity from the problem of full-dimensional clustering to that of subspace mining. They use a weighted Euclidean distance, so that constrained dimensions have larger weights. Due to the poor performance of the Euclidean distance function for high dimensions, this algorithm is expected to perform worse for mining highly-constrained subspaces\(^5\). Also, these methods use thresholds and parameters, which are hard to set intuitively and prescribe schemes, which are chosen more heuristically than theoretically.

\(^5\)Note however, that highly-constrained subspaces are unusual in practice.
2.1.1.2 Projected Clustering

Unlike the density-based subspace clustering methods, they do not prescribe an interestingness measure.

Aggarwal [4, 5] uses projective clustering to partition the dataset into clusters occurring in possibly different subsets of dimensions in a high dimensional dataset. PROCLUS [4] seeks to find axis-aligned subspaces by a three stage approach. First, a set of \( k \) cluster medoids are iteratively selected, without replacement, from the dataset, using a greedy approach. Second, a CLARANS-related[40], hill-climbing technique is used to cluster the data. In each iteration, the dataset is covered by spheres \( L_i \), centered at medoid \( m_i \). For each \( m_i \), a set of dimensions \( \mathcal{D}_i \), is determined for which, the points in \( L_i \) show little variation. The dataset is then partitioned using the nearest neighbor algorithm, with Manhattan distance computed from medoid \( m_i \) only over \( \mathcal{D}_i \). Finally, after the average intra-cluster Manhattan distance stabilizes, a cluster refinement phase is used.

ORCLUS [5], finds arbitrarily oriented clusters by using a variant of the iterative agglomerative hierarchical clustering. Initially \( k_e > k \) full dimensional points are randomly selected as subspace centers. In each iteration using the nearest neighbor algorithm, the distance of each point in the dataset to each \( d_e \)-dimensional subspace is determined and the dataset is partitioned. The eigenvectors corresponding to the smallest \( \beta d_e \) eigenvalues for the covariance matrix for the points in each resulting partition are retained and using merging, the number of clusters decreases by \( \alpha < 1 \). Both the algorithms require the number of clusters and the expected number of dimensions for each cluster to be input.

In DOC [68], Procopiuc et al. devise a Monte Carlo algorithm for finding projective clusters. They propose a mathematical formulation for the notion of optimal projective cluster based on the density of the points in the subspaces. A subspace is defined as a set \( C \) of points, including at least, fraction \( \alpha \) of the points in the dataset, spanning an interval of range no more than \( w \) in each constrained dimension and more than \( w \) for unconstrained dimensions. Via random sampling, they guess point \( p \) in the subspace and then for \( m \) iterations, a set \( X \) of \( r \) points, are randomly sampled from the dataset. They then find the set of dimensions
$D$, for which all points in $X$ are no more than $w$ apart from $p$. They then find the subspace for the set $D$ of constrained dimensions. Only $r$ and $m$ are chosen according to theoretical analysis.

In LDR [21], Chakrabarti et al. search for local correlations in the data and perform dimensionality reduction on the locally correlated clusters of data individually.

2.1.1.3 Biclustering

The problem of projected clustering is typically known as biclustering in the computational biology literature and there have been significant efforts to discover a wider range of patterns than those discussed in this dissertation. Microarray data contains the expression values for a set of genes/rows for a number of conditions/columns. A bicluster is a subset of genes/rows showing consistent expression patterns over a subset of the conditions.

Tanay et al. [80], discover statistically significant biclusters by posing the problem as one of finding heavily weighted bicliques, where the genes and conditions correspond to the vertices of the two halves of a bipartite graph. An edge exists between a gene and condition, if the gene is either significantly underexpressed or overexpressed under that condition. This approach cannot be easily extended to multinomial/categorical data.

Getz et al. [35], iteratively use a one-dimensional clustering algorithm to first cluster the set of genes using all the conditions into subsets, and then the set of conditions within each subset of genes into biclusters. They repeat this process until they get pairings between subsets of genes and conditions i.e., biclusters. This algorithm is greedy in nature and likely to miss overlapping subspaces.

A good review of the wide array of algorithms available in this field is available in Madeira et al. [57].

2.1.2 Finding relationships across the components of the model

The components of the model typically realize a partition of the dataset. To find the relationships between these partitions we need a distance or similarity measure. There exist a number of desirable properties for a distance measure $d$ between
components \( o \) and \( o' \). They are

1. Metric: i.e. it must satisfy the properties of:
   - positivity: \( d(o, o') \geq 0, d(o, o') = 0 \iff o = o' \)
   - symmetricity: \( d(o, o') = d(o', o) \)
   - triangle inequality: \( d(o, o') \leq d(o, o'') + d(o'', o') \)

2. Efficient to compute

3. Intuitive and understandable

4. Handle components that overlap in space, as well as those that do not.

Accordingly, there exist a number of types of similarity measures for comparing components of the model.

**Information theory-based similarity measures** Conventionally, datasets are compared by using information-theoretic measures such as the

1. Kullback-Leibler \((KL\) divergence [54]: The \( KL \) distance, also called relative entropy, is defined for discrete datasets \( D_A, D_B \) having identical schemas as

\[
KL(D_A : D_B) = \sum_{i=1}^{d_A} \sum_{k \in S_{A,i}} p_{k,A_i} \log \left( \frac{p_{k,A_i}}{p_{k,B_i}} \right)
\]

where \( p_{k,A_i}, p_{k,B_i} \) is the probability of finding the element \( k \) in the \( i \)th dimension of the dataset \( D_A, D_B \) respectively. The KL distance is a convex function and equals 0 if \( \forall i \in [1, d_A], k \in S_{A,i}, p_{k,A_i} = p_{k,B_i} \). However, it is unbounded, i.e., tends to infinity if \( \exists some k, i \) such that \( p_{k,B_i} = 0 \). It is an asymmetric dissimilarity measure.

2. J-distance/J-divergence [39]: Also called symmetric \( KL \) measure, it is defined as \( \eta(D_A : D_B) = KL(D_A : D_B) + KL(D_B : D_A) \)
Support-based similarity measures for clusterings  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 indices, Fowlkes-Mallows index, Rand index

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

the Jaccard index

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

Meila [58] proposes the VI (variance of information) metric to compare clusterings.

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

$$H(C) = \sum_{i=1}^{[C]} -p_i log(p_i) \text{ where } p_i = \frac{n_i}{n} \text{ and}$$

$$I(C, C') = \sum_{i=1}^{[C]} \sum_{j=1}^{[C']} p_{i,j} log\left(\frac{p_{i,j}}{p_i p_j}\right), \quad p_{i,j} = \frac{|C_i \cap C_j|}{n},$$

$n_i$ being the number of points in $C_i$, the $i$th 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 [40].

Statistic-based similarity tests  There are two types of similarity tests:
**Parametric similarity tests:** These similarity tests make assumptions about the underlying distribution of observed data. The book, Numerical Recipes in C [67], cites a number of statistical tests used to compare two one-dimensional data samples, e.g. the *student's t-test* is used to estimate the probability that the standardized difference in sample means is as large as it is for the given samples by chance, assuming they are drawn from distributions with the same means. If this probability is very low, the samples are probably drawn from distributions with different means, and thus it can be used to compare the two distributions on the basis of their means. Here, the underlying populations are assumed to have identical variances. Similarly, the *F-test* can be used to see if two datasets have variances that are significantly different. The *Chi-Square test* is used to determine if the two datasets having categorical/discrete attributes belong to the same distribution. The *Hotelling $T^2$ test* may be used for multivariate normal distributions. These methods cannot be used on datasets which are mixtures of multiple distributions.

**Non-parametric similarity tests:** These similarity tests make no assumptions about the underlying distribution of observed data. The *Wilcoxon Mann-Whitney Test* is used to test the null hypothesis that two populations have identical distribution functions against the alternative hypothesis that the two distribution functions differ only with respect to location (median), if at all. It can be generalized to the *Kruskal-Wallis test*, which is used to see if three or more samples are taken from populations having identical distribution functions against the alternative hypothesis that at least two of the samples differ only with respect to location (median), if at all. It is the analogue to the *F-test* used in analysis of variance. They can be used for ordinal datasets as well. The *Kolmogorov-Smirnov test* is also used to determine if the two datasets are generated from the same distribution. MANOVA methods [46] are more powerful and efficient than the *t-test* and are used for testing multivariate datasets.

The drawbacks of the above mentioned similarity tests is that

1. They only test if the two datasets have the same distribution and do not measure how and in what manner they are similar
2. They assume simultaneous access to the two datasets which may not be possible.

2.2 Finding similarities across condensed representations

In finding similarities across condensed representations, there have been three primary approaches.

2.2.1 Dataset similarity measures

In this approach, the authors typically assume the datasets are from the same attribute space and share the same schema. They then define a dataset similarity measure.

Ganti et al. [33] compare datasets by comparing their corresponding 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 linear transformations described in Sec. 1.3.

Parathasarathy et al. [66] 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

$$
\text{Sim}(D_A, D_B) = \frac{\sum_{z \in F_A \cap F_B} \max\{0, 1 - \alpha (\sup_{z,A} - \sup_{z,B})\}}{|F_A \cup F_B|}
$$
where \( 0 \leq \alpha \leq 1 \) is a the parameter, which scales the difference in the supports of the itemsets, common to both \( D_A \) and \( D_B \), \( \text{sup}_{x,A} \), \( \text{sup}_{x,B} \) is the fraction of rows of datasets \(|D_A| \) and \(|D_B| \) containing \( x \).

Li et al. [55] first propose a similarity measure for comparing datasets having the same schema. They represent each dataset by a set of MFIIs and define a variant of the mutual information between the MFI sets of two datasets \( A \) and \( B \) as

\[
I(A, B) = \sum_{i \in \text{MFI}(A), j \in \text{MFI}(B)} \frac{|i \cap j|}{|i \cup j|} \log(1 + \frac{|i \cap j|}{|i \cup j|}) \ast \min(|i|, |j|)
\]

Accordingly, they define the similarity across the datasets, normalized by the arithmetic mean of a variant of mutual information of each dataset with itself.

\[
\text{Sim}(A, B) = \frac{I(A, B) \ast 2}{I(A, A) + I(B, B)}
\]

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 across 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 \( \xi \) attributes of one dataset with those of the other. They use the similarity of supports of two attributes which remain unmapped from the two datasets, as a measure of similarity between the attributes themselves.

### 2.2.2 Graph matching

In graph matching, we seek a matching between the two input graphs \( G_A \), \( G_B \).

**Definition 3.** Consider a one-to-one mapping \( f : V_A \rightarrow V_B \), such that \((u, v) \in E_A \) iff \( \exists (f(u), f(v)) \in E_B \). Now if \(|V_A| = |V_B|\), then \( f \) is called an isomorphism and \( G_A \) is said to be isomorphic to \( G_B \). Else if \(|V_A| \neq |V_B|\), then \( f \) is called a subgraph isomorphism and \( G_A \) is said to be subgraph isomorphic to \( G_B \).

This problem is called exact graph matching. If \( G_A \), \( G_B \) are attributed graphs,
i.e., their vertices and edges have attributes, then an exact graph matching seeks a matching, which additionally ensures that the attributes of the matching vertices and edges are identical. *Attributed relational graphs* (ARGs) are graphs having attributes associated with both vertices as well as edges. Typically, isomorphisms/subgraph isomorphisms do not exist for ARGs, corresponding to real datasets, due to noise and distortions.

The problem of finding a non-bijective mapping between the input graphs is called *inexact graph matching*, a problem known [1] to be NP-complete. A non-bijective mapping implies that either the mapping is not onto, i.e., some of the vertices in both $G_A$ and $G_B$ may remain unmatched or the mapping is not one-to-one or both. Our research is concerned with finding one-to-one matchings. Thus, rather than looking for a subgraph isomorphism, we only seek to find mappings which are *unusually similar*. Doing so reduces the likelihood of overfitting the matching between $G_A$ and $G_B$. The argument is analogous to that of using local alignment versus global alignment between nucleotide sequences in the field of computational biology. To solve this problem, most of the approaches define an objective function to evaluate the matching between a pair of input graphs (for example, see Eq. 1.2), and present an algorithm to optimize this objective function. Many of the past approaches involve matching between labeled or discrete-attributed graphs. Like the solutions to many other NP-hard problems, graph matching algorithms may be categorized as *enumerative* and *optimization-based*. However, some of the methods appear to use a hybrid of both the approaches.

### 2.2.2.1 Enumerative/Search-based graph matching algorithms

These methods typically produce optimal solutions but require exponential time and hence are limited to use on problems of small size. Ullman [83] produced the classical graph isomorphism algorithm, which uses a backtracking search of the state space, enhanced by a lookahead function to prune the exploration. However, it was not resilient to noise and distortion [74].

Shapiro and Haralick [74] use the number of consistent subgraphs as a similarity measure. They find cliques in the two graphs and try and map them to each
other, minimizing the number of unmapped subgraphs.

To counter the problem of noise and occlusions (as discussed in Sec. 1.3, a family of error-tolerant graph matching algorithms were designed. Eshera and Fu [30] and Sanfeliu and Fu [69] generalized the concept of string edit distance to graph edit distance as a measure of comparing graphs. In computing the graph edit distance, costs are associated with addition, deletion and substitution of edges and vertices. As in string edit distance, the algorithm seeks to minimize the cost associated with the sequence of graph edit operations required to transform one graph into the other. Sanfeliu and Fu [69] use a graph grammar and Tsai and Fu [82] use error correcting codes enhanced by heuristics for speed to match ARGs.

Note that as with the string edit distance, it is difficult to decide what costs to associate with the graph edit operations. Bunke et al. [18, 19] showed the relationship between edit distance and the size of the maximal common subgraph and that the costs associated with insertion and deletion of vertices can be arbitrarily defined leaving only two parameters, viz., edge and vertex substitution costs, to be inferred.

Messmer and Bunke decompose graphs into their subgraphs in an effort to index them [60]. This idea was further developed by El-Sonbaty and Ismail [29] who represented each graph by its underlying subgraphs and defined the similarity between subgraphs from the graphs to be compared in terms of error-correcting distance. The distance between the pair of graphs is then defined as the minimum matching cost of the weighted bipartite graph constructed from the decomposed subgraphs.

DePiero [27] uses a function of the number of paths of variable length \( r \) from a pair of vertices from the two discrete weighted graphs to be matched, to measure the similarity between the vertices. The Dempster-Shafer method is used to find the final matching based on these similarity values.

Wong and You [90] define graph similarity via entropy by using probability distributions of attributes of vertices and edges. Boyer and Kak [16] use attribute deviations between graphs using the logarithmic conditional information to guide structural matching.
2.2.2.2 Optimization-based graph matching algorithms

Optimization-based graph matching algorithms may be either posed as discrete optimization algorithms or continuous optimization algorithms.

Continuous optimization algorithms: To handle occlusions, most algorithms use a null vertex [73] or use vertex (and relevant edge) deletions [69] without penalty, increasing robustness to clutter.

In their classical paper Christmas et al. [24] devised a probabilistic relaxation matching algorithm using the Bayesian framework, which aims to iteratively reconfigure mappings to maximize the aposteriori probability of the mapping, given the attributes associated with the vertices and edges of the graphs to be matched. This algorithm, unlike algorithms described thus far is capable of many-to-one mappings of vertices of a graph to those of another, by utilizing evidence from unary (associated with the vertices) as well as binary (associated with the edges) attributes of the graphs as well as prior information. They use a normal distribution to compute compatibility coefficients between vertices of the different graphs.

Wilson et al. [88] develop the exponential Bayesian consistency measure and shows its relationship to pre-existing similarity measures like Shapiro’s number of inconsistently matched cliques. They also show empirically that null labeling of vertices to handle occlusions does not work as well as using the exponential consistency measure to guide vertex and relevant edge deletion.

Umeyama [84] uses an eigendecomposition method to recover the permutation matrix that maximizes the correlation or overlap of the adjacency matrices for unweighted graphs of the same size. Carcassoni and Hancock [20], use a hierarchical spectral method for correspondence matching of point-sets to increase robustness to structural differences.

Almohamad and Duffuaa [3] used a linear programming approach to solve the graph matching problem, which was further extended by Van Wyk and Clark in their approximate least squares graph matching algorithm [91].

Luo et al. [56] formulates the problem of structural graph matching, in which only connectivity guides the matching, as a maximum-likelihood estimation problem.
Discrete optimization algorithms: Genetic algorithms were first used by Hervault et al. [41] for image matching. Cross et al. [26] uses the Bayesian consistency measure developed by Wilson et al. [88] as the function to test the fitness of the solutions in the population. The initial population of solutions is uniformly sampled from the feasible search space. Also, they generate the next generation via a crossover operator which combines matched subgraphs rather than creating new solutions by randomly selecting matched pairs from the current generation.

Tabu search, employed by Williams et al. [87], and simulated annealing [26] are some of the discrete optimization techniques used thus far for graph matching. Neural networks have been used for a number of graph matching applications including hand-printed Chinese character recognition [79], overlapped shape recognition [78], frontal face authentication [51].

Van Wyk et al. [92] try to solve the optimization problem \( \min_P(||A_A - P A_B P^T||) \) where \( P \) is a \( |V_A| \times |V_B| \) permutation submatrix. Van Wyk et al. [93] show that the problem of matching ARGs, each having \( r \) and \( s \) edge and vertex attributes respectively, can be formulated as

\[
\min_{\Phi} \left( \sum_{i=1}^{r+s} ||\text{vec}(A_1(i)) - \Phi \text{vec}(A_2(i))||^q \right)
\]

where \( q \in \{1, 2\} \), \( A_1(i), A_2(i) \) are the adjacency matrices of the \( i \)th attribute of the two graphs to be matched, \( \Phi = P \otimes P \) where \( P \) is a \( |V_{A_1}| \times |V_{A_2}| \) permutation submatrix. The algorithm has complexity \( O(n^4) \), where \( n = \max(|V_{A_1}|, |V_{A_2}|) \).

2.2.3 Flow-based algorithms

In matching vertex \( v_a \in V_A \) to some vertex \( v_b \in V_B \), it is imperative to consider the similarity of the structural positionings of \( v_a, v_b \) in \( G_A, G_B \) respectively, i.e., we are trying to structurally hash vertices. Merely considering the similarity of the unary attributes of \( v_a, v_b \) makes the algorithm susceptible to clutter and noise. Similarity between unary attributes of vertices corresponding to clusters from different datasets can be determined by using statistic-based methods referred to in Sec. 2.1.2.
One of the approaches to solving the graph matching problem is to combine the two input graphs \( G_A, G_B \), using some variation of the product graph \( G \), whose vertex set is \( V_A \times V_B \) and \(((u_a, u_b), (v_a, v_b)) \in E(G) \iff (u_a, v_a) \in G_A \) and \((u_b, v_b) \in G_B \). In such a scenario, the problem is reduced to finding the similarity between the vertices of \( G \).

This similarity may be defined recursively using the idea “Two vertices are similar if vertices they are related to are similar”. This idea is used explicitly or implicitly by a number of propagation algorithms ([59, 45, 86]) for a range of applications. The recursive definition causes similarity to flow from one vertex to the other. A fix-point computation algorithm is then used to compute the value of the recursive function in a top-down iterative fashion.

White et al. [86] seek to find the relative importance in a network. Given unweighted directed graph \( G(V, E) \), they seek to find \( I(t \in V|r \in V) \) where \( I : V \times V \rightarrow \mathbb{R} \) is the importance function. They do so using a number of techniques including weighted paths, i.e., \( I(t|r) = \sum_{k=1}^{P(r,t,k)} \lambda^{-|p_k|} \) where \( P(r,t,k) \) is the set of paths from \( r \) to \( t \) which are vertex and edge disjoint and having path length no greater than \( k \), \( |P(r,t,k)| \) is the cardinality of the set \( P(r,t,k) \), \( p_i \) is the \( i \)-th path in \( P \), \( |p_i| \) is the length of path \( p_i \) and \( 1 \leq \lambda \leq \infty \) is a scalar coefficient. This causes paths between \( r, t \) to have importance decaying exponentially as their length increases.

They investigate the use of Markov chains using \( I(t|r) = \frac{1}{\sum_{n=1}^{\infty} n f_{rt}^{(n)}} \), i.e., the inverse of the expected number of random steps taken until the first arrival at \( t \), starting from \( r \), where \( f_{rt}^{(n)} \) is the probability that the chain reaches \( t \) from \( r \) in exactly \( n \) steps. They also investigate variations of the HITS [50] and PageRank [64] algorithms.

Blondel et al. [14] 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

\[
S_{k+1} = w_BS_kw_A^T + w_B^TS_kw_A
\]

Note that this model does not assume that \( w_A \) and \( w_B \) are symmetric. This algo-
riothm has time complexity of matrix multiplication, which is currently \( O(n^{2.376}) \).

Melnik et al. [59], attempt to match unweighted, edge-labeled, directed graphs. To determine the matching between a pair of graphs, \( G_1(V_1, E_1, l_1) \) and \( G_2(V_2, E_2, l_2) \), where \( l_1, l_2 \) are labeling functions which take as parameter an edge in \( G_1, G_2 \) respectively; they create a pairwise connectivity graph \( PCG(V, E, l) \) of the graphs. Here, 

\[
((x, y), (x', y')) \in E, \quad l(((x, y), (x', y')) = p \iff (x, x') \in E_1, (y, y') \in E_2, \quad l_1(x, x') = l_2(y, y') = p,
\]

i.e., identically labeled edges in both graphs indicate that the source and target vertices are similar in that respect and hence are connected by an edge with that label in the PCG. They convert the PCG into an induced propagation graph, containing an additional edge going in the opposite direction for every edge in the PCG. Edges are weighted using function \( w : E \to \mathbb{R} \), so that for each type of label \( \forall v \in V, \sum_{x \in v} w(v, x) = 1.0 \). The similarity between any pair of vertices from \( G_1, G_2 \) is then a variable associated with each vertex in \( G \) and initialized to 1.0, i.e.,

\[
\sigma^0((x, y) \in V_1 \times V_2) = 1.0.\]

They then use this recursive equation to compute the steady-state similarities, i.e.,

\[
\sigma^{i+1}((x, y) \in V_1 \times V_2) = \\
\sigma^i(x, y) + \sum_{(a_u, x) \in E_1, (b_u, y) \in E_2, l_1(a_u, x) = l_2(b_u, y)} \sigma^i(a_u, b_u) \ast w((a_u, b_u), (x, y)) \\
+ \sum_{(x, a_v) \in E_1, (y, b_v) \in E_2, l_1(x, a_v) = l_2(y, b_v)} \sigma^i(a_v, b_v) \ast w((a_v, b_v), (x, y))
\]

2.3 Finding similarities between multiple datasets

The literature involving identification of similarities between multiple datasets stems primarily from the work done in solving the frequent subgraph discovery problem. A typical application of this problem is finding similar motifs/substructures occurring in a set of proteins/chemical compounds, improving storage efficiency of semi-structured databases, video indexing etc. The problem of frequent subgraph discovery has been studied primarily for vertex and edge-labeled graphs.

Among the first frequent substructure algorithms were AGM [44] in which Inokuchi et al. used an Apriori-based breadth-first search for frequent induced subgraphs. In gSpan [94], Yan et al. use a depth-first search (DFS) for candidate frequent subgraph generation. The depth-first traversal of the graph imposes a linear
ordering on the vertices based on the sequence in which they were discovered. In gSpan, a frequent subgraph $G$ is extended to a candidate frequent subgraph $G'$ by choosing a vertex $v$ in $G$ and adding a frequent edge $(v, w)$ where $w$ is not a vertex in $G$. Subgraph isomorphism checking is achieved by assigning each graph a DFS code.

Huan et al. [43], search for subgraphs having higher than a threshold frequency, i.e., frequent subgraphs, in the set of proteins belonging to the same structural and functional family in the SCOP database. Each protein has a dataset in which each row corresponds to the 3-d coordinates of an amino acid residue. They then represent each protein by an adjacency matrix in which each entry is either a 0 (if the Euclidean distance between the amino acid residues corresponding to its row and column is greater than a threshold value), a vertex label (if it is a diagonal element), or an edge label (if the Euclidean distance between the amino acid residues corresponding to its row and column is less than a threshold value). They define the code of this adjacency matrix as the sequence of the entries in the lower triangular matrix, read going left to right and then top to bottom. They use the lexicographic order to impose an ordering on the codes obtained by rearranging the rows of the matrix. Using a novel graph representation such as this, they construct a rooted, ordered tree for each graph and then search for frequent graphs. They then discard those which have a low mutual information score.

Computational biology approaches to mining via integration of diverse data sources include dataset tiling. For example, Tanay et al. [81], compiled a dataset including 70 sets of conditions for yeast ORFs from 27 experiments, by tiling information for the genes from the different experiments. Although many do, resulting clusters need not contain conditions from different experiments. Zhou et al. [95], in a manner similar to our method, first extract patterns from each dataset and then look for second order correlations among patterns from the different datasets. However, unlike our method, they directly compare the patterns from different datasets with each other, whereas we indirectly do so by comparing their interactions with other patterns, within their datasets. These approaches assume that a correspondence between rows from different datasets exists and is given, however.
CHAPTER 3
CONDENSED REPRESENTATIONS OF DATASETS

In the figures below, we outline the steps involved in generation of condensed representations from the input datasets. In Fig. 3.1, we see two datasets of points, represented as circles\(^6\), enclosed in a rectangle denoting the domains of the respective datasets. In Fig. 3.2, we apply our algorithm to find the subspaces, defined in

![Figure 3.1: The original sets of points in their own domains](image)

Sec. 1.2 and denoted by rectangles, in the datasets. Note that the subspaces are axis parallel. Also, the points in each subspace are assigned to that subspace. Thus, the points which remain uncovered, are labeled as noise or outliers. Finally, in Fig. 3.3, we determine the inter-relationships between the components/subspaces, i.e., the weighted adjacency matrices of the graphs corresponding to the datasets, denoted by the dashed lines between the subspaces. Thus, we generate a model for each dataset, consisting only of the subspaces and their inter-relationships. We delve into the details of the implementation of these concepts in the next two subsections.

\(^6\)Kindly disregard the thickness of lines/circles in these figures
3.1 Finding “interesting” subspaces in the dataset

We model the dataset using a set of possibly overlapping “interesting” subspaces. We have devised an algorithm called SCHISM (Support and Chernoff-Hoeffding based Interesting Subspace Mining) for mining interesting subspaces/components in the dataset. It uses the Chernoff-Hoeffding bound to prune the search for interesting subspaces as a nonlinear function of the number of dimensions in which the subspace is constrained. We use a vertical representation of the dataset and capitalize on various advances made in itemset mining[38]. We use a depth-
with backtracking to find maximal interesting subspaces. We start by discussing our interestingness measure.

3.1.1 Interestingness Measure

In addition to the notation specified in Sec. 1.2, we now introduce further notation required for the definition of our interestingness measure.

Let \( X_p \) be the random variable (RV) denoting the number of points in the \( p \)-subspace, \( S_p \). If the probability of finding \( n_p \) points in \( S_p \), is bounded by a reasonably low, user-specified threshold probability \( \tau \), \( S_p \) is considered to be interesting \(^7\), i.e., \( Pr(X_p \geq n_p) \leq \tau \) implies that \( S_p \) is an interesting subspace. Accordingly, we have the following definition of an interesting subspace:

**Definition 4.** A subspace is **interesting** if the number of points it contains is statistically significantly higher than that expected under the assumption that all dimensions are independent and uniformly distributed.

Note that such an “interesting” subspace implies that its constrained dimensions are positively (as the number of points it contains is larger than that expected) correlated (as it contradicts the assumption of independently distributed dimensions).

It is obvious that a dataset that is scattered uniformly and independently, spanning the entire dataset domain \( S \), is of little interest from a clustering viewpoint, as the entropy is maximized. If a subspace deviates significantly from the uniform distribution, then it is potentially interesting \(^8\). **CLIQUE** \(^8\) considers \( S_p \) to be ‘dense’, i.e., interesting, if \( n_p/n \geq s \), where \( s \) is the user-specified support threshold. **MAFIA** \(^62\) considers the subspace ‘dense’ if \( n_p/n \geq (aa)/D_i \), where \( a \) is the cluster dominance factor.

In general, all density-based subspace finding algorithms, use a threshold function \( \text{thresh} : Z^+ \rightarrow \mathbb{R} \) where \( Z^+ \) is the set of positive integers, and denotes the number of constrained dimensions in a candidate subspace. The value of \( \text{thresh} \in \mathbb{R} \) corresponds to the interestingness threshold that must be exceeded for the candidate

\(^7\)Typically \( \tau \) is set to \( O(1/n) \ll 0.05 \), which is statistically significant

\(^8\)See Eq. 3.3 and following comment
subspace to be called 'dense'. For example, for support based pruning in **CLIQUE**, 
\[ \text{thresh}_{\text{CLIQUE}}(p) = s, \forall p \in [1, d], \] 
where \( s \) is the minimum support threshold described in Sec. 2.1.1.1. This implies that as shown in Fig. 3.4, no matter what the number of constrained dimensions of a subspace, the pruning threshold is a constant. This is counterintuitive, in that as \( p \) increases, the volume of the \( p \)-subspace decreases exponentially, and hence the expected number of points in it should also decrease. This causes **CLIQUE** to have a bias favoring \( p \)-subspaces, where \( p \ll d \).

![Figure 3.4](image)

**Figure 3.4:** Effect of the number of constrained dimensions on \( \text{thresh} \)

The \( \text{thresh} \) function can intuitively be either constant (as in **CLIQUE**) or monotonically increasing or monotonically decreasing.

**Lemma 1 (Effect of \( \text{thresh} \) on monotonicity):**

*If any given subspace \( S_{p+1} \subset S \) is interesting, then every \( p \)-subspace \( S_p \), which encloses \( S_{p+1} \) and is unconstrained in one of the \((p + 1)\) constrained dimensions of \( S_{p+1} \), is always interesting if \( \text{thresh}(p + 1) \geq \text{thresh}(p), 1 \leq p \leq d - 1 \), for interestingness threshold function \( \text{thresh} \).*

**Proof:** If \( S_{p+1} \) is interesting, \( \frac{n_{p+1}}{n} \geq \text{thresh}(p + 1) \). But, \( n_p \geq n_{p+1} \) because \( S_{p+1} \subset S_p \). Thus, \( \frac{n_p}{n} \geq \frac{n_{p+1}}{n} \geq \text{thresh}(p + 1) \). If, \( \text{thresh}(p + 1) \geq \text{thresh}(p) \), then \( \frac{n_p}{n} \geq \text{thresh}(p) \) and we are guaranteed to find all interesting subspaces. \( \square \)

**Lemma 2:** For \( p=1\ldots d \), let \( \text{thresh} \) be monotonically increasing and let \( \text{thresh}_2(p) = \text{thresh}(r) \), where \( r \) is the number of dimensions constrained in \( S_r \), a maximal inter-
esting subspace under thresh. If \( S_r \subseteq S_p \), then \( S_p \) is also interesting under \( \text{thresh}_2 \).

**Proof:** Since \( S_r \) is interesting, then \( \frac{n_r}{n} \geq \text{thresh}(r) \). Also, as \( S_r \subseteq S_p, 1 \leq p \leq r - 1 \), \( n_p \geq n_r \). Hence, \( \frac{n_p}{n} \geq \frac{n_r}{n} \geq \text{thresh}(r) = \text{thresh}_2(p) \). Thus, \( S_p \) is also interesting under \( \text{thresh}_2 \). \( \Box \)

A consequence of the above lemmas is that it in order to find all the maximal interesting subspaces found by \( \text{thresh}_2 \), one must set \( \text{thresh}(1) \) to a very low value. Hence, it makes little sense to have a monotonically increasing threshold function. The \( \text{thresh} \) function must be either constant or monotonically decreasing. However, it has been observed, that in order to find small subspaces, the constant support threshold function has to be set very low. This makes subspace mining very slow. Hence, we propose a non-linear monotonically decreasing threshold function as shown in Fig. 3.4, i.e., SCHISM, which does not guarantee mining all interesting subspaces,\(^9\) but does mine in a more reasonable time. The intuition behind why this might work, is that as \( p \) increases, the subspace becomes constrained in more and more dimensions, making its volume smaller and smaller. Hence the threshold too, must decrease for the enclosed \( (S_{p+1}) \) and enclosing \( (S_p) \) subspaces to have comparable interestingness.

### 3.1.2 Chernoff-Hoeffding bound

We use the Chernoff-Hoeffding bound \([23, 42]\) to bound the tail of the distribution of \( X_p \), introduced in Sec. 3.1.1 and measure the level of interestingness. If \( Y_i, i = 1 \ldots n \), are independently distributed RV, with \( 0 \leq Y_i \leq 1 \) and \( \text{Var}[Y_i] < \infty \), then for \( Y = \sum_{i=1}^{n} Y_i, t > 0, \)

\[
Pr[Y \geq E[Y] + nt] \leq e^{-2nt^2}
\]

(3.1)

where \( E[Y] = \sum_{i=1}^{n} E[Y_i] \) by linearity of expectation.

Given a \( p \)-subspace \( S_p \), let \( Y_i \) correspond to the RV that the \( i \)th point in \( DB \), when projected onto the set of \( p \) constrained dimensions of \( S_p \), lies within \( S_p \). Then

\(^9\)For monotonically decreasing \( \text{thresh} \), the Apriori principle, i.e., if \( S_p \) is dense, then every \((p+1)\)-subspace enclosed by it, is dense, may not be applicable. Consider, \( \frac{n_r}{n} = \frac{n_{p+1}}{n} = \text{thresh}(p + 1) \). If \( \text{thresh}(p + 1) < \text{thresh}(p) \), then \( \frac{n_r}{n} < \text{thresh}(p) \) and \( S_p \) is not interesting although \( S_{p+1} \) is.
Y = X_p. Using Eq. (3.1) and for some real \( t_p > 0 \), \( S_p \) is interesting if,

\[
Pr[X_p \geq n_p] \leq e^{-2nt_p^2} \leq \tau
\]

(3.2)

where \( E[X_p] + nt_p = n_p \), which implies that \( t_p = \frac{n_p}{n} - \frac{E[X_p]}{n} \). Substituting \( t_p \) in the right hand term of (3.2), we have \( e^{-2n\left(\frac{n_p}{n} - \frac{E[X_p]}{n}\right)^2} \leq \tau \) which on simplification gives,

\[
\frac{n_p}{n} \geq \frac{E[X_p]}{n} + \sqrt{\frac{1}{2n} \ln\left(\frac{1}{\tau}\right)}
\]

(3.3)

Thus, for a \( p \)-subspace to be interesting, (3.3) must hold. (3.3) makes no assumption, other than independence, about the comparative distribution and hence can be extended to find subspaces interesting w.r.t. distributions, other than that having uniformly distributed dimensions.

Note that the interestingness measure, \( \text{thresh}(p) = \frac{E[X_p]}{n} + \sqrt{\frac{1}{2n} \ln\left(\frac{1}{\tau}\right)} \) is a non-linear monotonically decreasing function in the number of dimensions \( p \), in which \( S_p \) is constrained. Also, note that \( \text{thresh} \) is analogous to the support and density threshold measures used to prune search in the CLIQUE [8] and MAFIA [62] algorithms respectively. In comparison with CLIQUE, the term \( \sqrt{\frac{1}{2n} \ln\left(\frac{1}{\tau}\right)} \) corresponds to minimum density \( s \) set by the user. The interestingness threshold probability (\( \tau \)) seems intuitively easier to set than \( s \). The chief difference is the term \( \frac{E[X_p]}{n} \), which makes pruning conscious of the volume of the subspace and hence conscious of the number of constrained dimensions of the subspace on which it is being carried out. Unlike earlier proposed interestingness measures [47], this one gives the user a sense of absolute interestingness.

If we assume that each dimension in the \( d \)-dimensional space is independent and uniformly distributed and discretized into \( \xi \) levels, then the probability that a point lies in a specific interval of any dimension is \( \frac{1}{\xi} \). Hence, the probability that a point lies in a specific \( p \)-subspace (assuming it is constrained to a single interval in each of the \( p \) constrained dimensions) is \( (\frac{1}{\xi})^p \). Thus, the probability of finding \( n_p \) points in any subspace \( S_p \), is distributed as per the binomial distribution with mean \( E(X_p) = n\left(\frac{1}{\xi}\right)^p \). Thus, \( S_p \) is interesting if, \( \frac{n_p}{n} \geq \frac{1}{\xi^p} + \sqrt{\frac{1}{2n} \ln\left(\frac{1}{\tau}\right)} \).
Note that for \( p \geq \left\lfloor \frac{\log(n)}{\log(\xi)} \right\rfloor = v \), we have \( \frac{n}{\xi^p} \leq 1 \), and thus \( \frac{1}{\xi^p} + \sqrt{\frac{1}{2n} \ln \left( \frac{1}{\tau} \right)} \approx \sqrt{\frac{1}{2n} \ln \left( \frac{1}{\tau} \right)} \). The threshold function thus converges to a constant when the number of constrained dimensions \( p \geq v \); analogous to minimum threshold \( s \) in \textsc{CLIQUE} as shown in Fig. 3.4. To summarize,

\[
\text{thresh}_{\text{SCHISM}}(d \geq p \geq v) = \sqrt{\frac{1}{2n} \ln \left( \frac{1}{\tau} \right)}
\]

From Lemma 1, for \( p \geq v \), \( S_{p+1} \) is interesting implies that \( S_p \) is interesting, as \textsc{SCHISM} is similar to \textsc{CLIQUE} and uses support-based pruning for a large part of the subspace mining process. Note that this constant \( \sqrt{\frac{1}{2n} \ln \left( \frac{1}{\tau} \right)} \) varies inversely as \( \sqrt{n} \) and hence \( \text{thresh}_{\text{SCHISM}} \) converges to a higher threshold for smaller datasets, implying more pruning while yielding subspaces of equal interestingness.

While \( \text{thresh}_{\text{SCHISM}} \) is constant for \( p \geq v \), we can gain some improvements in empirical results by changing the rate of change in \( \text{thresh}_{\text{SCHISM}}(p < v) \) to increase the likelihood of monotonic search. We do so by trading off some tightness of the bound by using a penalty term. If \( f(p) \) is the penalty term, such that \( \forall p \in [1, d], f(p) \leq 1 \), then \( e^{-2n_f(p)} \leq e^{-2nf(p)^2} \). Using this in 3.2, \( \Pr(X_p \geq n_p) \leq e^{-2nf(p)^2} \leq \tau \). After simplification,

\[
\text{thresh}_{\text{SCHISM}}(0 < p < v) = \min \left\{ u, \frac{1}{\xi^p} + \sqrt{\frac{1}{2nf(p)} \ln \left( \frac{1}{\tau} \right)} \right\}
\]

The term \( u \) is used to upper bound \( \text{thresh}_{\text{SCHISM}}(1) \), which is empirically too large for typical values of \( \xi \). One possible value of \( f(p) \) is the scaled penalty term, i.e., \( a \times p \). Here \( a \geq v \) implies that \( f(p) \leq 1 \) and \( a \) is a user-specified constant. Alternatively, we may use the parabolic penalty term, i.e., \( \frac{1}{(c-bp^2)} \), which requires parameters \( b \) and \( c \). The last penalty term provides a parabolic as opposed to exponential drop in the threshold as \( p \) increases. Typically, \( u = 0.05 \). In summary, we have

\[
\text{thresh}_{\text{SCHISM}}(p) = \begin{cases} 
\min \left\{ u, \frac{1}{\xi^p} + \sqrt{\frac{1}{2nf(p)} \ln \left( \frac{1}{\tau} \right)} \right\} & \text{if } p < v \\
\sqrt{\frac{1}{2n} \ln \left( \frac{1}{\tau} \right)} & \text{if } p \geq v
\end{cases}
\]  

(3.4)
To mine more subspaces than those found by the support threshold \((u)\) in CLIQUE [8],

\[
u \geq \frac{1}{\xi v} + \sqrt{\frac{1}{2n} \ln \left( \frac{1}{\tau} \right)}\]  

(3.5)

Substituting typical values e.g., \(\tau = 10^{-5}, n = 10^5, \xi = 10, v = 5\) in Eq. (3.5), we get \(u \geq 0.0076\). It is unusual to set \(u\) to lower values for typical high-dimensional datasets, as that would make Apriori-based search virtually impossible. Thus, using \(thresh_{SCHISM}\) helps mine for smaller subspaces than CLIQUE.

3.1.3 The SCHISM Algorithm

A number of the subspace mining algorithms [8, 22, 62] use a bottom-up, breadth-first search. In contrast, SCHISM, which is based on the GenMax algorithm that mines maximal itemsets [38], uses a depth-first search with backtracking to mine the maximal interesting subspaces. It also performs several optimizations such as using diffsets and progressive focusing [38]. The main steps in SCHISM are

\[
\begin{align*}
\text{SCHISM} \ (DB, s, \xi, \tau): \\
//s \text{ is the minimum support threshold} \\
//\xi \text{ is the number of intervals per dimension} \\
//\tau \text{ is the user-specified interestingness threshold} \\
1. & \ DDB = \text{Discretize}(DB, \xi) \\
2. & \ VDB = \text{HorizontalToVertical}(DDB) \\
3. & \ MIS = \text{MineSubspaces}(VDB, s, \xi, \tau) \\
4. & \ \text{AssignPoints} \ (DB, MIS)
\end{align*}
\]

Figure 3.5: The SCHISM Algorithm

shown in Fig. 3.5; we first discretize the dataset and convert it to a vertical format. Then we mine the maximal interesting subspaces. Finally, we assign each point to its cluster, or label it as an outlier.

**Discretization:** In SCHISM, we first discretize all points (figure 3.5, line 1). Given the original dataset \(DB\), we divide each dimension into \(\xi\) bins, and give each interval a unique id (for example, the intervals in dimension \(d_0\) are labeled from 0 to
\[
\begin{array}{|c|cccccccc|}
\hline
DB & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 \\
\hline
p_1 & 755 & 689 & 306 & 482 & 838 & 657 & 743 & 980 \\
p_2 & 818 & 166 & 494 & 302 & 378 & 439 & 633 & 805 \\
p_3 & 418 & 159 & 499 & 260 & 139 & 921 & 986 & 780 \\
p_4 & 833 & 173 & 484 & 236 & 948 & 17 & 647 & 781 \\
p_5 & 264 & 960 & 465 & 985 & 70 & 209 & 782 & 309 \\
p_6 & 991 & 972 & 118 & 986 & 72 & 209 & 804 & 341 \\
p_7 & 921 & 963 & 910 & 976 & 71 & 220 & 818 & 317 \\
p_8 & 686 & 965 & 623 & 993 & 68 & 202 & 800 & 287 \\
p_9 & 448 & 146 & 605 & 205 & 984 & 423 & 654 & 983 \\
\hline
\end{array}
\]

(a) Original DB

\[
\begin{array}{|c|cccccccc|}
\hline
DDB & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 \\
\hline
p_1 & 7 & 16 & 23 & 34 & 48 & 56 & 67 & 79 \\
p_2 & 8 & 11 & 24 & 33 & 43 & 54 & 66 & 78 \\
p_3 & 4 & 11 & 24 & 32 & 41 & 59 & 69 & 77 \\
p_4 & 8 & 11 & 24 & 32 & 49 & 50 & 66 & 77 \\
p_5 & 2 & 19 & 24 & 39 & 40 & 52 & 67 & 73 \\
p_6 & 9 & 19 & 21 & 39 & 40 & 52 & 68 & 73 \\
p_7 & 9 & 19 & 29 & 39 & 40 & 52 & 68 & 73 \\
p_8 & 6 & 19 & 26 & 39 & 40 & 52 & 68 & 72 \\
p_9 & 4 & 11 & 26 & 32 & 49 & 54 & 66 & 79 \\
\hline
\end{array}
\]

(b) Discretized DB

**Table 3.1: Example DB: Real & Discretized**

\(\xi - 1\), those for \(d_1\) are labeled from \(\xi\) to \(2\xi - 1\), etc.). Consider the example dataset \(DB\) shown in Table 3.1 (a), generated by our synthetic data generator (see section 5.2.1), with number of points in dataset, \(n = 9\), number of divisions per dimension, \(\xi = 10\) and number of dimensions in dataset space, \(d = 8\). The seed subspaces used
<table>
<thead>
<tr>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
<th>$d_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>478</td>
<td>-1</td>
<td>673</td>
<td>774</td>
</tr>
<tr>
<td>$I_2$</td>
<td>-1</td>
<td>163</td>
<td>475</td>
<td>260</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$I_3$</td>
<td>-1</td>
<td>949</td>
<td>-1</td>
<td>985</td>
<td>72</td>
<td>204</td>
<td>806</td>
</tr>
</tbody>
</table>

(a) Original Subspaces

<table>
<thead>
<tr>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
<th>$d_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>34</td>
<td>-1</td>
<td>56</td>
<td>67</td>
</tr>
<tr>
<td>$I_2$</td>
<td>-1</td>
<td>11</td>
<td>24</td>
<td>32</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>$I_3$</td>
<td>-1</td>
<td>19</td>
<td>-1</td>
<td>39</td>
<td>40</td>
<td>52</td>
<td>68</td>
</tr>
</tbody>
</table>

(b) Discretized Subspaces

Table 3.2: Seed Subspaces: Real & Discretized

to generate $DB$ are shown in Table 3.2 (a). Here $-1$ implies that the subspaces are unconstrained in that dimension. Thus, $p_1$, i.e., [755 689 306 482 838 657 743 980], is generated from subspace $I_1$, which is constrained to a multinormal cluster in the dimensions $\{d_6, d_7, d_8\}$ centered at [478 673 774] respectively. Similarly, points $p_2, p_3, p_4$ are generated from $I_2$, points $p_5, p_6, p_7, p_8$ are generated from $I_3$ and $p_9$ is an outlier. Table 3.1 (b) shows the discretized dataset $DDB$ obtained from $DB$; the corresponding discretized subspaces are shown in Table 3.2(b).

Data Transformation: The next step in SCHISM (figure 3.5, line 2) is to convert the dataset into a vertical tidset format [38], which records for each subspace (initially a single interval), the list of points that belong to it. Using a vertical format dataset gives us a number of advantages. Firstly, better memory utilization results from having only the relevant subspaces in memory at a time, as opposed to the horizontal format in which the entire dataset is scanned. Secondly, computing support of subspaces to be merged via tidset intersections is very fast. Fig. 3.7 (for $p = 1$) shows the tidsets for the initial ‘interesting’ intervals. For example, for interval 11 corresponding to interval [100,200] of dimension $d_2$, its tidset is given as
$t(11) = \{2, 3, 4, 9\}$.

**Mining Interesting Subspaces:** In SCHISM, interesting subspaces are mined (figure 3.5, line 3), using a depth-first search with backtracking, allowing us to prune a considerable portion of the search space. The pseudo-code for MineSubspaces is shown in Fig. 3.6. The method first finds all interesting subspaces in one ($IS_1$) and two dimensions ($IS_2$). We next call the recursive MIS-backtrack procedure to mine the set of maximal interesting subspaces ($MIS$).

**MIS-backtrack** accepts as input, a single $p$-subspace $S_p$, and a set $C_p$ of candidate $p$-subspaces that can be used to constrain (or extend) $S_p$ in an interval of another dimension. Each $S_x \in C_p$ results in a potential new $p+1$-subspace $S_{p+1} = S_p \cup S_x$ (line 5), for which we have to calculate the new candidate set $C_{p+1}$ (line 8). We do this by using a possible set $P_{p+1}$ (line 6) of potential candidate subspaces, which are all the unprocessed subspaces in $S_y > S_x$ in $C_p$. If $C_{p+1}$ is empty (line 9), then $S_{p+1}$ is potentially maximal; it will be added to MIS if there is no maximal subspace that encompasses it (lines 10-11). If $C_{p+1}$ is non-empty, then we recursively call MIS-backtrack.

The call to IS-Candidate (line 8) constructs the new candidate set $C_{p+1}$ for $S_{p+1}$ for the next level. The basic idea is to intersect tidset of $S_{p+1}$ with every possible subspace in $P_{p+1}$ (line 20). We keep only those subspace extensions that pass the thresh() function (lines 21-22).

Typically, a depth-first search with backtracking produces a number of subspaces, which may overlap considerably, leading to redundant subspaces. To avoid this behavior, we prune the search tree by using MergeSubspaces (line 7). If the subspace $Z_{p+1} = S_{p+1} \cup P_{p+1}$, resulting from constraining $S_{p+1}$ with all its remaining possible intervals in $P_{p+1}$, is significantly similar (typically, we set the merging threshold $\rho = 0.8$) to some known $S_f \in MIS$ (line 13), we replace $S_f$ with a typically more constrained subspace (lines 14-15), $S_f \cup Z_{p+1}$. 
MineSubspaces \((VDB, \xi, \tau)\):
1. Find \(IS_1\) and \(IS_2\) //sort \(IS_1\) as optimization
2. MIS-backtrack\((\emptyset, IS_1, MIS, 0, \xi, \tau)\)
3. return \(MIS\)

MIS-backtrack\((S_p, C_p, MIS, l, \xi, \tau)\)
4. \(\forall S_x \in C_p\)
5. \(S_{p+1} = S_p \cup S_x\)
6. \(P_{p+1} = \{S_y \in C_p | S_y > S_x\}\)
7. If MergeSubspaces\((MIS, (S_{p+1} \cup P_{p+1}))\) return
8. \(C_{p+1} = IS\text{-candidate}(S_{p+1}, P_{p+1}, l, \xi, \tau)\)
9. If \(C_{p+1}\) is empty
10. If \(S_{p+1}\) has enclosed no subspace in \(MIS\),
11. \(MIS = MIS \cup S_{p+1}\)
12. else MIS-backtrack\((S_{p+1}, C_{p+1}, MIS, p + 1, \xi, \tau)\)

MergeSubspaces\((MIS, Z_{p+1})\)
13. If \(\min_{S_f \in MIS} Sim(Z_{p+1}, S_f) > \rho \times \min(f, |Z_{p+1}|)\) return
14. \(MIS = MIS - S_f\)
15. \(MIS = MIS \cup (S_f \cup (Z_{p+1})\)
16. return true
17. return false

IS-candidate\((S_{p+1}, P_{p+1}, l, \xi, \tau)\)
18. \(C_{p+1} = \emptyset\)
19. \(\forall S_y \in P_{p+1}\)
20. \(t(S'_y) = t(S_{p+1}) \cap t(S_y)\)
21. If \(\frac{|t(S'_y)|}{n} \geq \text{thresh}(|S'_y|)\) return \(C_{p+1}\)
22. \(C_{p+1} = C_{p+1} \cup S_y\)
23. return \(C_{p+1}\)

Figure 3.6: Mining Interesting Subspaces
As $\rho < 1$, we may merge subspaces, which are constrained to adjacent intervals in a few dimensions, thus compensating for uniform width intervals in each dimension. The $Sim$ function (line 13) used to calculate the similarity of two subspaces $A$ and $B$ is given as

$$Sim(A, B) = \sum_{i=1}^{d} JaccardSimilarity(A_i, B_i)$$

where $A_i, B_i$ are the sets of interesting intervals spanned by $A, B$ in the $i$-th dimension and

$$JaccardSimilarity(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

For example for discretized point, $p_1 \in DDB$, i.e., [7 16 23 34 48 56 67 79] in Table 3.1(b), and discretized seed subspace $I_2$, i.e., [24 32 77] in Table 3.2)(b), $Sim(p_1, I_2) = 2$, as they are identically constrained in the second (11) and third (24) dimensions.

**Example** Let's consider how SCHISM works on our example dataset $DB$ (see Fig. 3.7). Let $u = 0.25$ and the interestingness threshold, $\tau = 4/n = 0.44$. Then, $IS_1 = \{11, 19, 24, 32, 39, 40, 52, 68, 73\}$. Likewise we compute $IS_2$. The initial call to MIS-backtrack is with $S_p = \emptyset$, and $C_p = IS_1$, which results in a recursive call of MIS-backtrack for each interval, with a new candidate set $C_{p+1}$. For example, the candidate-set for 11 is given by $C_1 = \{24, 32, 39, 40, 52, 68, 73\}$, thus in the next level, we will try to extend 11 with 24, 32, $\cdots$, 73, and so on, recursively. For our running example, when $S_{p+1} = 11, S_y = 24$, then $S'_y = \{11, 24\}$. Also $t(S_{p+1}) = \{2, 3, 4, 9\}$, and $t(S_y) = \{2, 3, 4, 5\}$, which gives $t(S'_y) = \{2, 3, 4\}$ (see Fig. 3.7). As $p = 1$, we use the interestingness measure for pruning the search (line 21), i.e., the second case in Equation 3.4. With $thresh_{SCHISM}(1) = 0.21, n_1/n = 3/9 = 0.33 > 0.22$; thus $S'_y$ is interesting, and we add 24 to the candidate set. Proceeding in this manner, we get the lattice shown in Fig. 3.7. The rectangles shaded in gray are the elements of MIS.
Figure 3.7: Lattice of running example

**Assigning Points to Clusters** Based on the intervals to which each subspace is constrained, we can estimate discrete probability distribution functions (p.d.f.) for each dimension for each mined subspace. If each dimension of the $d$-dimensional dataset is discretized into $\xi$ equi-width intervals, then $B(i, j)$ corresponds to the probability, that subspace $B$ is constrained in the $j$th interval of the $i$th dimension. Thus, $\forall i \in [1, d], \sum_{j=1}^{\xi} B(i, j) = 1.0$. If $b(i)$ is the number of intervals to which $B$
AssignPoints \((DB, MIS)\):

1. \(\forall i \in [1, n] \) // for each point in \(DB\)
2. If \(\max_j Sim(x_i, MIS_j) > ThresholdSim\)
3. \(x_i \rightarrow MIS_{\text{argmax}_j Sim(x_i, MIS_j)}\)
4. else \(x_i\) is an outlier

**Figure 3.8: Assigning Points in SCHISM**

is constrained in dimension \(i\),

\[
\forall i \in [1, d], B(i, j) = \begin{cases} 
\frac{1}{b(i)} & \text{if } B \text{ is constrained in dimension } i \\
0 & \text{otherwise}
\end{cases}
\]  

(3.6)

Let \(A\) correspond to the \(d\)-subspace surrounding a point \(p_i\). Let \(Y\) be the RV denoting the similarity between \(A\) and \(B\), and \(y\) be the true similarity between \(A\) and \(B\), respectively.

We define similarity between two subspaces as the sum of the dot products of their p.d.f.s, across all dimensions. Accordingly,

\[
Sim(A, B) = \sum_{i=1}^{d} Y_i, \quad Y_i = \sum_{j=1}^{\xi} A(i, j) B(i, j)
\]

Then, \(E(Y_i) = 1 \times \frac{1}{b(i)} \times \frac{\xi}{\xi} = \frac{1}{\xi}\). \(Y\) is then, the sum of \(d\) Bernoulli RVs with mean \(E[Y] = \frac{d}{\xi}\). Using Chernoff-Hoeffding bounds again, if \(Pr[Y \geq y] \leq \exp(-2t^2) \leq \tau\) for reasonably small user-specified threshold \(\tau\), it implies that the similarity between \(A\) and \(B\) is unusually high and the point in \(A\) is with high probability generated from the subspace \(B\). Now, \(\exp(-2t^2) \leq \tau\) implies that \(t \geq \sqrt{\frac{1}{2d} \ln \left(\frac{\xi}{\tau}\right)}\). Substituting \(t\) in \(y = E[Y] + dt = \frac{d}{\xi} + dt\), we get \(ThresholdSim = \frac{d}{\xi} + \sqrt{\frac{d}{2} \ln \left(\frac{\xi}{\tau}\right)}\).

Fig. 3.5, line 4 of SCHISM assigns each point to the most similar maximal interesting subspace (lines 2-3), or else labels it as an outlier (line 4). Fig. 3.8 shows these steps. Additionally, we have to examine if the similarity is statistically significant by using \(ThresholdSim\) computed above.
3.2 Representing the dataset as a graph

Each dataset $D_i$ can be represented by a graph $G_i(V_i, E_i, w)$ where $V_i$ is the set of subspaces found by SCHISM and $w : E_i \rightarrow \mathbb{R}$ is a similarity function used to weight the edges of $G_i$. Depending on whether we use support or similarity of projections as the basis for comparing subspaces, we prescribe the following subspace similarity measures:

3.2.1 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 $S_A\setminus u$. Accordingly, if $C_u, C_{u'}$ are the clusterings yielded by subspaces $u, u' \in V_A$, we can define $w_A(u, u')$ using $Jaccard(C_u, C_{u'})$ and $Rand(C_u, C_{u'})$ as described in Sec. 2.1.2. Additionally, we experiment with using $w_A(u, u') = \exp(-VI(C_u, C_{u'}))$.

3.2.2 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 Sec. 3.2.1, 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

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

(3.7)

Here, $Q_i$, $R_i$ refer to the set of intervals spanned by subspaces $Q$, $R$ respectively, in dimension $i$. If dimension $i$ of $R$ is unconstrained, $|R_i| = \xi$. Using our running example from Sec. 3.1.3, $\rho(p_1 \in g_1, c_1) = \frac{1}{d_A} \left( \frac{1}{\xi} + \frac{1}{\xi} + \frac{1}{\xi} + \frac{1}{\xi} + \frac{1}{\xi} \right) = 0.28$

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:

### 3.2.2.1 AVGSIM

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 Eq. 3.7, the similarity between two subspaces \( u \in V_A, u' \in V_A \) can be defined as

\[
    w_A(u, u') = \frac{\sum_{r \in u} \rho(r, u')}{|u|} + \sum_{r \in u'} \frac{\rho(r, u)}{|u'|} \tag{3.8}
\]

From our running example in Sec. 3.1.3, \( \rho(p_1 \in g_1, c_2) = 0.24, \rho(p_2 \in g_2, c_2) = 0.44, \rho(p_3 \in g_3, c_1) = \rho(p_4 \in g_4, c_1) = \rho(p_5 \in g_5, c_1) = 0.28 \). Then, \( w_A(c_1, c_2) = \frac{0.24 + 0.44 + 0.28 + 0.28}{3} = 0.62 \). \( \rho(r, R) = \sum_{i=1}^{d_i} \frac{1}{R(i)} \) To ensure that \( \forall u \in V_A, w_A(u, u) = 1 \), we normalize by setting \( w_A(u, u') = \frac{w_A(u, u')}{\sqrt{w_A(u, u) \times w_A(u', u')}} \)

### 3.2.2.2 HIST

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 \( d_{D_i} \)-dimensional dataset is discretized into \( \xi \) equi-width intervals, then \( u(i, j) \) corresponds to the fraction of points assigned to vertex/subspace \( u \) discretized to the \( j \)th interval in the \( i \)th dimension. Accordingly,

\[
    w_A((u, v) \in E_A) = \frac{1}{d_{D_i} \xi} \sum_{i=1}^{D_i} \sum_{j=1}^{\xi} \text{sim}(u(i, j), v(i, j))
\]

where \( \text{sim} : [0, 1] \times [0, 1] \rightarrow [0, 1] \) is a similarity function. We have tested a number of similarity functions:

1. Dot product: \( \text{sim}(a, b) = a \cdot b \)

2. Histogram: \( \text{sim}(a, b) = \frac{\min(a, b)}{\max(a, b)} \)
3. Gaussian weighted: \( \text{sim}(a, b) = \exp\left(\frac{-(a-b)^2}{2s^2}\right) \)

4. Increasing weighted: \( \text{sim}(a, b) = \frac{1}{1+\left|\frac{a-b}{s}\right|} \)

5. Sigmoidal weighted: \( \text{sim}(a, b) = 2 \times \log(\cosh(\frac{2\times|a-b|}{s})) \)

where \( s \) is a user-defined parameter controlling the spread of \( \text{sim} \). Note that for \( u = v \in V_A \) and using HIST and Gaussian weighted \( \text{sim}() \), \( w_A(u, v) = 1.0 \).
CHAPTER 4
FINDING SIMILARITIES ACROSS CONDENSED REPRESENTATIONS OF PAIRS OF DATASETS

To illustrate the functionality realized in this chapter, we continue with the running graphical example, introduced at the start of Chapter 3. Our input (see Fig. 4.1) to this stage is a pair of graphs, embedded in their respective domains. The output of the algorithms described in this chapter is a matching between vertices from the two graphs, shown by solid bidirectional arrows (see Fig. 4.2).

![Figure 4.1: Input: graphs for each dataset using the found subspaces](image)

From the example, it is evident that the dataset on the left has been subjected to a 90 degree clockwise rotational transformation to produce the dataset on the right, i.e., the Y-axis in the dataset on the left hand side corresponds to the X-axis of the dataset on the right hand side.

To find similarities across the graphs, we test four algorithms. One (Monte-Carlo Match) uses MonteCarlo sampling (see Sec. 4.1). Another (OLGA) uses the tensor product, another Gibbs sampling, and the last (RandomMatch) uses randomization followed by refinement.
4.1 MonteCarlo Match

One way of estimating the unusualness of matchings produced by our algorithms, involves generating random matchings and comparing the err (as defined in Eq. 1.2) value of the best of these, as defined in Eq. 1.2, with that produced by our algorithms. This is known as MonteCarlo Match/MC Match.

Accordingly, if $|V_A| \geq |V_B|$, 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 10,000 times, evaluate them using the Zscore described in Sec. 5.1.2 and keep the one with the best Zscore.

4.2 OLGA

In this algorithm, we tackle the problem of matching all the vertices in the smaller graph to some subset of the vertices in the larger graph. We do so, by estimating the similarities between pairs of vertices from the two graphs using structural information. Once this structural similarity between pairs of vertices is estimated, we retain the set of mutually disjoint pairs of vertices having high structural similarity. The latter step is modeled by combining the vertices of the graphs to be matched, viz. $G_A$ and $G_B$, into a single bipartite graph $G = (V_A \cup V_B, E \subseteq V_A \times V_B, \Pi_{A,B})$. 
$\Pi_{A,B}$ is a $|V_A| \times |V_B|$ matrix of structural similarities between pairs of vertices from the two graphs, which we need to estimate.

To find $\Pi_{A,B}$, we construct the product graph (see function \texttt{ProductGraph} in Figure 4.3), $G' = (V_A \times V_B, E' \subseteq (V_A \times V_B) \times (V_A \times V_B), w_{A,B})$, where $w_{A,B} : E' \rightarrow \mathbb{R}$ is the adjacency matrix, indicating similarity between vertices corresponding to pairs of subspaces from underlying graphs, of $G'$.

\[
 w_{A,B}((u, v), (u', v')) = \begin{cases} 
 \text{sim}(w_A(u, u'), w_B(v, v')) & \text{if } \text{sim}(w_A(u, u'), w_B(v, v')) > \tau \\
 0 & \text{otherwise}
\end{cases}
\]  

(4.1)

where $\tau$ is a user-specified threshold, used to minimize noise and limit space complexity of the algorithm. As $w_A(u, u')$, $w_B(v, v')$ depend on $G_A, G_B$ respectively, the weight of an edge in product graph $G'$, 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 $S = \text{vec}(\Pi_{A,B})$ (using notation from Sec. 1.2), a $l = |V_A||V_B|$ length column vector. 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 use a fix-point iterative algorithm to compute $S$. Accordingly, if $S_i$ denotes $S$ at iteration $i$ of the algorithm, we can write this as

\[
 S_i((u, v)) = \sum_{u' \in V_A, v' \in V_B} w_{A,B}((u, v), (u', v')) S_{i-1}((u', v'))
\]

\[
 = w_{A,B}((u, v), :) \cdot S_{i-1}
\]

Then, $S_i = w_{A,B} \cdot S_{i-1}$

where $w_{A,B}((u, v), :)$ returns the $(u, v)$th row of $w_{A,B}$. As shown in Figure 4.3, we set the initial similarities, i.e., all entries in $S_0$ to 1.0 (line 6). We then iterate using Eq. 4.2 (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 $\epsilon$ (line 9).
ProductGraph\((G, G_A, G_B)\):
1. \(\forall (u, v) \in (V_A \times V_B)\) create vertex \((u, v)\)
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. 4.1

OLGA\((G_A, G_B, \tau, k)\):
5. ProductGraph\((G, G_A, G_B)\)
6. \(S_0 = \text{ones}(|V_A|, |V_B|)\)
7. for i=1:k
8. \(S_i = \frac{w_{A,B}S_{i-1}}{\|w_{A,B}S_{i-1}\|_F}\)
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. \(\Pi_{A,B} = U_{A,1} \otimes U_{B,1}^T\)
15. return Match\((\Pi_{A,B})\)

Figure 4.3: OLGA, FastOLGA: 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., \(\Pi_{A,B}\).

Ideally, \(\Pi_{A,B}\) is a permutation submatrix which minimizes \(err(f|w_A, w_B)\) (Eq. 1.2). Typically however, \(\Pi_{A,B}\) is a real matrix. Hence, we need to round \(\Pi_{A,B}\) to a permutation submatrix. We use the \textbf{Match} function to do the same. \textbf{Match} returns \(f: V_A \rightarrow V_B\), i.e., it maps vertices in \(G_A\) to those on \(G_B\). There are a number of matching algorithms, e.g., stable matching, the Kuhn-Munkres algorithm [53], perfectionist egalitarian polygamy [59], 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 [53]. 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 similarity propagation, in [59]. However, they use directed, labeled graphs.

If $w_{A,B}$ is normal, it is diagonalizable. Using the Gaussian weighted $\text{sim}()$ function introduced in Sec. 3.2.2.2 makes $w_{A,B}$ normal and hence diagonalizable. If it has a dominant eigenvalue,

$$S_i = \frac{w_{A,B} \cdot S_{i-1}}{||w_{A,B} \cdot S_{i-1}||_F}$$

Then, $S' = \lim_{i \to \infty} S_i = \frac{w_{A,B} \cdot S'}{||w_{A,B} \cdot S'||_F}$ (4.2)

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 $\frac{\lambda_2}{\lambda_1}$ [37].

If $\text{sim}$ 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 $w_{A,B} = w_A \otimes w_B$, using notation from Sec. 1.2. 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 4.3.

It is known [85] 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., $w_{A,B} = w_A \otimes w_B = 1 \leq i, j \leq |V_A|, |V_B|, \lambda_{w_{A,i}}\lambda_{w_{B,j}}$ 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 $S' = \text{vec}(U_{A,1} \otimes U_{B,1})$, using notation from Sec. 1.2 for dominant eigenvectors. Finding a maximal weighted 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 the dominant eigenvectors.

The dominant eigenvector of an $n \times n$ matrix can be determined in $O(n^2)$ time (lines 11,12) using QR factorization [37] 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, which has complexity $O(|V_A|^2|V_B| + |V_A||V_B|^2)$.

### 4.3 Mapping via Gibbs sampling

Often, while matching graphs, some vertices may ideally not have any vertex matched with them, e.g., occlusions in computer vision. Using Gibbs sampling, Neuwald et al. propose techniques [63], which can be adapted to enhance graph matching.

We use Gibbs sampling [63] to minimize the function $err$ in Eq. (1.2). First, we estimate the background distribution (lines 1-5). To do so, we generate a set $F$ of matchings (line 1), using MonteCarlo (MC) sampling as described in Sec. 4.1 (typically $|F| = 1000$). We use $F$ as a sample of the matching space to estimate the background distribution of the objective function. For each such matching $f \in F$, 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., $f'_{u,v} = (f \setminus (u, f(u))) \cup (u, v)$. We define the score of mapping $u$ to $v$, written as, $s(u \rightarrow v)$, as a generalization of the Gaussian weighted function given in Sec. 3.2. Then, $s(u \rightarrow v)$ is the negative exponential of the mean error incurred in the set of
matchings \( F'(u, v) = \{ f' : f'(u) = v \} \), i.e.,

\[
s(u \rightarrow v) = \exp\left( -\sum_{f' \in F'(u,v)} \frac{|f'| \\text{err}(f')}{|F'(u,v)|} \right)
\]

(4.3)

Having estimated the background distribution, we generate seed mappings (line 8). As the computational complexity of Gibbs sampling is high, we restrict the algorithm to 100 seed mappings. For each seed, as in the background estimation stage, we estimate the score of mapping \( u \) to \( v \), i.e., \( q(u, v) \) using \( \exp(-|f| \text{err}(f \setminus (u, f(u)) \cup (u, v))) \). Then we sample from \( v \in V_B \), using distribution \( a(u, v) \), which is proportional to the ratio of the seed and background distributions, i.e.,

\[
a(u, v) \propto \frac{q(u, v)}{s(u \rightarrow v)} \quad \text{Then, } a(u, v) = \frac{\frac{q(u, v)}{s(u \rightarrow v)}}{\sum_{x \in V_B} \frac{q(u, x)}{s(u \rightarrow x)}}
\]

(4.4)

If this ratio is large, it indicates that the seed matching has small error w.r.t. the background distribution and hence it is more likely to be selected. 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 RandomMatch

As opposed to our earlier work [72] in this field, where we proposed algorithms for matching all subspaces in one dataset to a subset of subspaces in the other, we use a randomized approach as shown in Fig. 4.3. We create a number of seed matchings and then refine them to match more vertices, thereby some vertices in both graphs may remain unmatched.

Determining the number of seed matchings: Let \( \Psi \subseteq V_A \) be vertices in \( G_A \), which are optimally matched to some \( |\Psi| \) vertices in \( G_B \). If we randomly pick \( \Phi \subseteq V_A \) (line 2), a set of \( g \) vertices from \( G_A \), \( g = |\Phi| < |\Psi| \), then the probability
**GibbsSampling**\((G_A, G_B)\):

1. for \(i = 1:1000\)
2. generate MC matching \(f\)
3. \(\forall u \in V_A\)
4. \(\forall v \in V_B\)
5. estimate \(s(u, v)\) using Eq. 4.3
6. MinGlobalError = \(\infty\)
7. for \(i = 1:100\)
8. generate MC matching \(f\)
9. MinLocalError = \(\infty\)
10. for \(i = 1:1000\)
11. \(\forall u \in V_A\)
12. \(\forall v \in V_B\)
13. \(q(u, v) = \exp(-|\text{err}(f \setminus (u, f(u)) \cup (u, v)))\)
14. pick \(v\) from \(a(u, v)\), update \(f\) to \(f'\)
15. MinLocalError = min\((\text{err}(f'), \text{MinLocalError})\)
16. MinGlobalError = min\((\text{MinGlobalError}, \text{MinLocalError})\)

**Figure 4.4: Gibbs sampling**

that \(\Phi\) is a subset of \(\Psi\) is given by the hypergeometric distribution as

\[
P_r[\Phi \subset \Psi] = \frac{{|\Psi|\choose g}{|V_A| - |\Psi|\choose 0}}{{|V_A|\choose g}} = \frac{{|\Psi|\choose g}}{{|V_A|\choose g}}
\]

Hence, the probability that at least one of \(t\) such random pickings of \(\Phi\) (line 1), is completely covered/contained by \(\Psi\) is given by the geometric distribution as \(1 - (1 - P_r[\Phi \subset \Psi])^t\). We then upper bound this probability, with a reasonable user-specified threshold, \(\delta\) (typically 0.03). Thus, we can determine the number of trials \(t\) needed by solving \(1 - (1 - P_r[\Phi \subset \Psi])^t \leq 1 - \delta\).

**ProjectionMatch**: As shown in Fig. 4.5, we only use the subgraph \(G_{A'}\), induced by the set of vertices in \(\Phi\) to produce a matching with \(G_B\) (line 3). Thus, the problem
RandomMatch\((g, G_A, G_B)\):

1. for \(i=1:t\)
2. pick a \(g\)-subset, \(\Phi \in V_A\)
3. \(G_{A'}\) is induced from \(G_A\) using \(\Phi\)
4. \(P = \text{ProjectionMatch}(G_{A'}, G_B)\)
5. while not converged
6. \(P' = \text{PruneSeed}(P, G_A, G_B)\)
7. \(P = \text{ExtendSeed}(P', G_A, G_B, g)\)
8. display \(P\)

ProjectionMatch\((G_{A'}, G_B)\):

9. Find \(U_{A'}, D_{A'}, U_B, D_B\)
10. \(X_{A'} = U_{A'}D_{A'}^{1/2}, X_B = U_BD_B^{1/2}\)
11. Project \(X_B\) onto \(d_{A'}\) principal axes to get \(X_{B'}\)
12. \(\forall (i, j) \in [1, d_{A'}] \times [1, d_{A'}]\)
13. \(\Pi_{A', B}(i, j) = \exp(- (X_{A', i} - X_{B', j})^2)\)
14. return \(\text{Match}(\Pi_{A', B})\)

Figure 4.5: RandomMatch: Matching two subgraphs

is reduced to one of finding a subgraph in \(G_B\) to match to \(G_{A'}\). In function ProjectionMatch, we minimize the optimization function \(err\) in Eq. (1.2) in a manner related to that of Scott and Longuet-Higgins [70]. We wish to \(\min_P err(f|w_{A'}, w_B)\)

\[
\rightarrow \min_P ||w_{A'} - Pw_BP^T||_F^2 \quad \text{as} \quad g = |f| \quad \text{is constant}
\]

But, \(w_{A'} = X_{A'}X_{A'}^T\) as \(w_{A'}\) is positive semidefinite (p.s.d)

Also, \(w_{A'} = U_{A'}D_{A'}U_{A'}^T\) by eigendecomposition of \(w_{A'}\)

Thus, \(X_{A'} = U_{A'}D_{A'}^{1/2}\)

Similarly, \(w_B = X_BX_B^T = U_BD_BU_B^T\) and \(X_B = U_BD_B^{1/2}\)

\[
\Rightarrow \min_P ||X_{A'}X_{A'}^T - PX_BX_B^TP^T||_F^2
\]
Now, if $X_{A'} = PX_B$, $err$ is minimized. This is impossible if $X_{A'}$ and $PX_B$ are not of the same size. Hence, we introduce matrix $Q$, which is a $d_{A'} \times d_B$ matrix, to project $PX_B$ to the same space as $X_{A'}$. Thus we solve,

$$\min_{P,Q} \|X_{A'} - PX_BQ^T\|_F^2 + \|X_BQ^TQX_B^T - w_B\|_F^2$$

as an approximation to the original problem. Now, the right-hand term is minimized if $Q$ is a permutation submatrix which selects the $d_{A'}$ largest eigenvalues from $X_B$ (line 11). If we use this value of $Q$, then the problem of finding $P$ is identical to that of bipartite weighted matching, and is solvable using the Hungarian algorithm [53], which has earlier been discussed in Sec. 4.2.

**PruneSeed:** In **PruneSeed**, we record the error resulting when each mapped pair is individually dropped (line 4) from the set of matched vertices. If a pair of matched vertices are dropped and this results in a large drop in the error, it implies that the pair are not a good match structurally. To define a large drop in error, we make use of the commonly used definition of an outlier, while examining box-plots, i.e., if the error is less than the mean error by more than 1.5 times the inter-quartile range, i.e., $Q_3 - Q_1$ (line 6), the corresponding pair of vertices are removed from the matching (line 7).

**ExtendSeed:** In **ExtendSeed**, we seek to extend the local mappings surrounding each pair of mapped vertices (line 10). For example, if $u \in V_A$ maps to $v \in V_B$, we seek to extend the mapping between the local neighborhoods of $u$ and $v$. We first obtain lists of unmapped vertices among the k-nearest neighbors (k-NN) of $u$ and $v$ (lines 11,14). Then, using the already mapped vertices, which are among the k-nearest neighbors of $u$ and $v$ (lines 12, 15), we choose pairs of unmapped vertices from these lists which produce the least increase in error, if added to the match (lines 13, 16) and update the matching (line 17).

For all the refinement algorithms, we define convergence as the point at which the seed does not change.
\textbf{PruneSeed}(P, G_A, G_B):
1. while not converged
2. \(E = \emptyset\), i.e., empty set
3. \(\forall u \in V_A\), s.t. maps to some vertex \(v \in V_B\)
4. \(E = E \cup err(f \setminus (u, f(u)))\)
5. \(\forall u \in V_A\), s.t. maps to some vertex \(v \in V_B\)
6. if \(E(u) < Q_2 - 1.5 \times (Q_3(E) - Q_1(E))\)
7. \(f = f \setminus (u, f(u))\)
8. return \(P(f)\)

\textbf{ExtendSeed}(P, G_A, G_B, k):
9. while not converged
10. for each mapped vertex \(m \in V_A\)
11. \(\forall u : u\) is an unmapped k-NN of \(m\)
12. retain mapping \(f'\), involving NN of \(u\)
13. \(c1 = (u, \text{argmin}_{v \in V_B} err(f' \cup (u, v)))\)
14. \(\forall v : v\) is an unmapped k-NN of \(f(m)\)
15. retain mapping \(f'\), involving NN of \(v\)
16. \(c2 = (\text{argmin}_{u \in V_A} err(f' \cup (u, v)), v)\)
17. \(f = f \cup \text{argmin}_{c \in \{c1, c2\}} err(f \cup c)\)
18. return \(P(f)\)

Figure 4.6: PruneSeed, ExpandSeed: Refining the match
CHAPTER 5
EXPERIMENTS & RESULTS

Having described the theory involved in finding similarities across high dimensional datasets in chapters 3 and 4 we now perform experiments on both synthetic and real datasets to evaluate the accuracy of our models and to provide a comparison between the different algorithms, both proposed and in the existing literature.

5.1 Performance measures

We propose different measures for the tasks of determining the components within the datasets and that of finding similarities across the models for the datasets.

5.1.1 Performance measures for condensed model representation

In determining the performance of our algorithm for identification of components in a dataset, we use the following measures:

1. Entropy: For a clustering $C$, entropy is defined as

   $$E(C) = \sum_{C_j} \frac{n_j}{n} E(C_j) = -\sum_{C_j} \frac{n_j}{n} \sum_i p_{ij} \log(p_{ij})$$

   where $p_{ij} = \frac{n_{ij}}{n}$, $C_j$ is the $j$th cluster in $C$, $n_j$ is the number of points in $C_j$, and $n_{ij}$ is the number of points of $C_j$ that actually belong to subspace $i$. From the formula, the entropy of a clustering is the sum of the entropies of the individual clusters, each weighted by the fraction of points assigned to the respective clusters. A low entropy clustering then implies that the clusters to which a large number of points are assigned have low entropy, i.e., they are relatively “pure”. Entropy is strictly non-negative and it reaches a minimum value at 0. The lower the $E(C)$, the purer the clusters and hence the better the clustering.

2. Coverage: It is defined as the fraction of points in $DB$, which are assigned to
some cluster in the clustering \( C \). By definition, the coverage of a clustering lies in \([0,1]\). Ideally, it is 1.

3. Execution time

There is a tradeoff between minimizing the entropy and maximizing the coverage. As the coverage increases, i.e., more points in the dataset are assigned to some cluster, the likelihood of error in assignment and hence the entropy increases and vice versa. Although, our measures do not include a factor to account for model complexity, i.e., the number of clusters/subspaces produced by our algorithm, our algorithm \textbf{MergeSubspaces} in Fig. 3.6 typically ensures that the trivial case, in which each point is assigned to its own subspace, is never reached.

5.1.2 Performance measures for finding similarities across pairs of condensed models

In determining the performance of our algorithms, we shall pay attention to the following measures:

1. \( Z\text{score} \): If \( w_A, w_B \) are the matrix of pairwise vertex similarities in graphs \( G_A, G_B \) respectively, and \( G_B \) has at least as many vertices as \( G_A \), then a one-to-one matching \( f \) yields matrix \( P_f \) having score \( err(f|w_A, w_B)/|f| \) using the definition of \( err \) from Eq. 1.2). \(|f|\) denotes the size of the matching, i.e., the number of nodes matched in either graph. We estimate the distribution of this function by generating a number of matchings using MonteCarlo sampling (see Sec. 4.1) and computing the corresponding \( errs \). Using this distribution, the mean and standard deviation of \( err \) can be determined. The \( err \) corresponding to a mapping found by an algorithm is standardized to get the \( Z\text{score} \). Thus, the \( Z\text{score} \) is the number of standard deviations from the mean. Very negative \( Z\text{scores} \) imply the corresponding matching is very unlikely to have happened by MonteCarlo sampling (Sec. 4.1) and such a matching is said to have found \textit{unusually similar} substructure.

2. Number of matches, i.e., \( #(matches) \): It is the number of \( D_B \)'s matchable components that are correctly matched. A component in \( D_B \) is \textit{matchable} if
there exists a known, unusually similar component in $D_A$.

3. Execution time

As the $\#(matches)$ measure requires the knowledge of the matchable components, it is typically used during verification of accuracy on labeled data and is unsuitable for unlabeled data. Also, note that by applying Chebyshev’s inequality to the Zscore of a matching, an upper bound for the p-value of that matching may be obtained. In

![Histogram of Zscores for random matchings between 8-vertex and 21-vertex graphs](image)

**Figure 5.1: Histogram of Zscores for random matchings between 8-vertex and 21-vertex graphs**

Fig. 5.1, we plot the histogram of the Zscores of 10,000 random matchings between 8 vertices of one graph and 21 of another yielded from our synthetic datasets. It is evident from the graph, that the Zscores for the synthetic datasets appear to be Gaussian distributed.
5.2 Datasets

In our experiments, we wish to evaluate our algorithms by testing its performance as a function of the number of

- dimensions in the datasets
- interesting subspaces in the datasets
- matching interesting subspaces shared by the datasets
- points in the datasets
- on homogeneous (all continuous/discrete attributes) and heterogeneous datasets
- on datasets having noise and nulls

5.2.1 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 [71]. It has the following set of parameters:

1. Average number of dimensions \((d)\)
2. Average number of points in a dataset \((n)\)
3. Probability that a point is noisy \((r)\)
4. Average number of embedded subspaces \((k)\)
5. Probability that a subspace is constrained in a dimension \((c)\)
6. Probability that a subspace is constrained in the same dimension as the previous subspace, i.e., the consecutively generated subspaces overlap in the set of dimensions, they are constrained to \((o)\)
7. Amount of perturbation \((p)\)
8. Type of transformation

We first generate parameters for 1.5$k$ subspaces and then insert a random subset of these subspaces into the datasets to be generated. Thus, in *SetSubspaceParameters* (see Fig. 5.2, we generate the parameters of 1.5 $k$ normally distributed subspaces. Every subspace is constrained within a $D$-dimensional hypercube, each dimension having range $[0,1000)$. By using variables $c_l, o_l$ which are uniformly distributed about $c, o$ respectively (line 2), we ensure that the subspaces have different volumes (line 5) and have different probabilities of overlapping (line 8) respectively.

```
SetSubspaceParameters(D, n, r, k, c, o):
1. for l = 1:1.5k
2. \( c_l = U[0.5, 1.5] \times c, o_l = U[0.5, 1.5] \times o \)
3. for d = 1:D
4. \( \sigma_l[d] = 20 \)
5. if \( U[0, 1) < c_l \)
6. if \( \mu_l[d] = -1 \)
7. \( \mu_l[d] = U[0, 1000) \)
8. if \( U[0, 1) < o_l \) and \( d < D \)
9. \( \mu_l[d + 1] = U[-1, 1) \times \sigma_l[d] + \mu_l[d] \)
```

**Figure 5.2: Set parameters for subspaces to be embedded**

The subspaces are multivariate normal with means in dimension \( d \in [1, D] \) chosen from $U[0,1000)$ (line 6), where $U[l,h)$ implies a uniform distribution over the interval $[l,h)$. The standard deviation in each dimension, i.e., $\sigma(d \in [1, D])$, is set to 20 (line 4). A dimension is constrained with average probability $c$. Two serially generated subspaces are constrained in the same dimension with average probability $o$. Their means are constrained to be within 2 standard deviations of each other (line 9), to allow overlapping of subspaces.

For $i \in \{1, 2\}$, for dataset $D_i, n_i$ and $k_i$ 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 per-
turing their parameters using a transformation. There are three types of transformations:

1. Noisy: $\forall d \in [1, D], \mu(d) = \mu(d) + U(-p, p) \times 1000, \sigma(d) = \sigma(d) \times (1 + U(-p, p))$

2. Translation: $\mu(d) = \mu(d) + i \times p \times 1000$

3. Scaling: $\sigma(d) = \sigma(d) \times (1 + ip/5)$

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 $a$, $a = \frac{\max_i \alpha_i}{\min_i \alpha_i}$ where $\alpha_i$ is the fraction of $D_i$ generated using parameters of the $l$th 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 $U[0,1000)$.

We use the following algorithm to generate similar datasets each having possibly overlapping subspaces in a high dimensional space.

In **GenerateDatasets** (see Fig. 5.3, for each dataset (line 2), we generate a uniformly distributed number of points, noise and subspaces (line 3). The interval which spans the uniform distribution is arbitrarily chosen. We then perturb the center ($\mu_i$) and standard deviation ($\sigma_i$) for each model cluster $i$ (lines 4-12). By having a variable number of subspaces in each dataset and by ensuring that they all have the first few generated from the same model, we can ensure that the datasets do have some similarity. Lines 13-15 generate noisy points.

In experiments for synthetic datasets shown below, on average, the datasets have, unless otherwise mentioned, $n = 1000$ points and $d = 50$ dimensions and $k = 25$ embedded subspaces, except those with $k > 40$ subspaces, which have $n = 10000$ points. Also, $c = o = 0.5$, $p = 0.03$, $a = 4.0$. We use the noisy transformation and Gaussian weighted $sim()$ function.

### 5.2.2 Real datasets

GenerateDatasets($\Sigma, D, n, r, k, c, o$):
1. SetSubspaceParameters($D, n, r, k, c, o$)
2. for $i = 1: \Sigma$
3. $n_i = rand(0.5, 1.5 \times n, r_i = rand(0.75, 1.25 \times r, k_i = rand(0.5, 1.5 \times k$
4. for $l = 1:k_i$
5. for $d = 1:D$
6. if $\mu_l[d] = -1$, continue
7. $\mu_l[d] = rand(0.5, 1.5) \times \mu_l[d]$
8. $\sigma_l[d] = rand(0.5, 1.5) \times \sigma_l[d]$
9. for $b = 1:f_i \times n_i$
10. for $d = 1:D$
11. if $\mu_l[d] = -1$, $x[i][d] = rand(0, 1000)$
12. else $x[i][d] = \mu_l[d] + \sigma_l[d] \times rand(0, 1)$
13. for $b = 1:r_i \times n_i$
14. for $d = 1:D$
15. $x[i][d] = rand(0, 1000)$

Figure 5.3: Generate datasets by embedding subspaces

the digit set [0,9] on a pressure-sensitive tablet. Each vector corresponds to the
(x, y) coordinates of 8 points, spatially sampled from each of these handwritten
digits. Note that the embedded subspaces, i.e., the digits 0-9, overlap considerably in
the 16-dimensional space.

National Basketball Association (NBA) dataset: We use the statistics of
players from the NBA, averaged annually from two consecutive basketball sea-
sons, viz., 2003-04 (dataset A) and 2004-05 (dataset B). They are accessible from
http://sports.yahoo.com/nba/stats/. Datasets A and B contain statistics for
443 and 464 players respectively. Each dataset has 16 columns, viz., number of games
played, average minutes played per game, average field goals, 3-pointers and free
throws made and attempted, offensive and defensive rebounds, assists, turnovers,
steals, blocks, personal fouls and points per game.
**Breast cancer datasets:** We use two datasets pertaining to breast cancer [12, 89] donated to the UCI ML repository at 


obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The first dataset (X) has 9 dimensions/columns having integral values between 1 and 10 for clump thickness, uniformity of cell shape and size, marginal adhesion, bare nuclei etc. and 699 samples/rows. There are a few missing values as well. 35% of the samples are malignant (M) and the rest are benign. The second dataset (Y) has 30 dimensions/columns corresponding to three statistics (mean, standard error, max) for each of ten real-valued features (radius, symmetry, area, texture, etc.) of the cell nuclei drawn from 569 samples/rows. Thus, the schema for X and Y are different. In Y, 37.25% are malignant and the rest are benign.

**Yeast microarray datasets:** In our specific application, we look at three microarray datasets, called GDS38, GDS124 and GDS39, pertaining to the *Saccharomyces cerevisiae* (yeast) cell cycle. The datasets are accessible via

http://www.ncbi.nlm.nih.gov/projects/geo/gds/gds_browse.cgi. The datasets contain expression values of the different genes of yeast sampled over its cell cycle. The culture are synchronized by different mechanisms, viz., alpha factor block-release (GDS38), cdc15 block release (GDS124) and centrifugal elutriation (GDS39). GDS38 has 16 samples/columns taken at 7 minute intervals, while GDS39 has 14 samples/columns taken at 30 minute intervals and GDS124 has 25 samples/columns taken from almost 3 full cell cycles. Datasets GDS38 and GDS39 have 7680 genes/rows, while GDS124 has 8832 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.

### 5.3 Results of condensed representations of datasets

We run the algorithm **SCHISM**, described in Chapter 3, on synthetic and real datasets described in Sec. 5.2. Our experiments for **SCHISM** were implemented
in C++ on a SunSparc 650 MHz machine running on a Solaris O/S with 256 MB RAM.

5.3.1 On synthetic datasets

Unless otherwise mentioned, we use $k = 5$, i.e., 5 subspaces embedded per dataset. We do not show the entropy results for SCHISM on the synthetic datasets, as they are typically below 0.04, i.e., SCHISM reports very pure clusters for the synthetic datasets. For the graphs corresponding to the performance of SCHISM on synthetic datasets, we use the term “Score” on the y-axis to denote different performance measures based on the context. Typically, it corresponds to the running time (in minutes) and the coverage.

5.3.1.1 Effect of varying dataset parameters

The first set of experiments on synthetic datasets deal test the performance of SCHISM as parameters used in generating the synthetic datasets, viz. $n, d, \kappa, c, o, k$ are varied. We also attempt varying the density and distribution, i.e., Gaussian or uniform, of the embedded subspaces. Finally, we vary the distribution of the background noise (Gaussian v/s uniform random).

![Figure 5.4: Effect of dataset size ($n$) on SCHISM's performance](image)


Figure 5.5: Effect of data dimensionality ($d$) on SCHISM’s performance

**Effect of dataset size and dimensionality:** In Fig. 5.4, it is evident that as the dataset size increases, the coverage remains constant, while the running time grows linearly. Note that in Fig. 5.5, as the dimensionality of the dataset increases, the coverage remains more or less constant, but the running time seems to grow exponentially initially, and then grows linearly from dimensions 200-300. In the worst case, this algorithm has exponential complexity in the number of dimensions, but in practice as shown here, the DFS algorithm coupled with the varying threshold function, significantly prunes the search space.

**Effect of subspace size and dimensionality:** In Fig. 5.6, we observe the variation in coverage and running time, as the ratio $\kappa = \frac{\text{max}_i \alpha_i}{\text{min}_i \alpha_i}$ increases from 2 to 12. We observe that as the ratio increases, the coverage dips marginally and the running time remains constant. The coverage decreases because the subspaces which contain a smaller number of points have, on average, as large a volume as those containing a larger number of points, leading to a lower density. A smaller fraction of their enclosing subspaces are likely to be identified as ‘interesting’ and hence only a small fraction of their points are detected as non-outliers, as compared to when the ratio is not so large.

In Fig. 5.7, we observe the variation in coverage and running time, as the
probability of constraining a dimension in a subspace $c$, increases from 0.3 to 0.9. We observe that as $c$ increases, the running time remains constant but larger fractions of the dataset are constrained to smaller volumes, making them more ‘interesting’ and hence coverage improves somewhat.

**Figure 5.7: Effect of subspace dimensionality ($c$) on SCHISM’s performance**

**Effect of subspace overlap:** Subspaces may overlap in terms of the dimensions in which they are constrained or in terms of the specific intervals they are con-
Figure 5.8: Effect of fraction of overlapping constrained dimensions (o) on SCHISM’s performance

strained to. Accordingly, we have two experiments. In the first, we test the effect on performance due to increased overlap between constrained dimensions of subspaces generated consecutively. By increasing o from 0.1 to 0.9, we increase the likelihood of different subspaces being constrained to the same intervals of the same dimensions. In Fig. 5.8, we observe that running time stays constant but coverage decreases as o increases. This occurs because it becomes more likely that points belong to multiple ‘interesting’ subspaces simultaneously and hence only one is discovered, which may not completely cover the other.

In the second experiment, if $C_{i,j}$ is the $j$th co-ordinate of the center of the $i$th subspace, then if we constrain the $j$th co-ordinate of the $(i+1)$th center as well, we set it so $C_{i+1,j} \in \{C_{i,j} - 2\sigma_{i,j}, C_{i,j} + 2\sigma_{i,j}\}$, where $\sigma_{i,j}$ is the standard deviation in the $j$th dimension of the points corresponding to the $i$th subspace. From Fig. 5.9, we observe that SCHISM does not perform too well in this test and fails to find some of the subspaces as the dimensionality rises, again because a point may now belong to multiple subspaces.

Effect of number of embedded subspaces $k$ for different background distributions In Fig. 5.10 and Fig. 5.11, we compare the performance of SCHISM as the background distribution switches from Gaussian noise to uniform random
Figure 5.9: Effect of constraining means in adjacent embedded subspaces to within 2σ

noise. For each type of background distribution, we plot the running time in minutes (in Fig. 5.10) and the entropy of the clustering (in Fig. 5.11) as the number of embedded subspaces (k) is varied. We do not show the coverage as it is 1. In Fig. 5.10, we note that finding subspaces against uniform noise takes less time than that for Gaussian noise. From Fig. 5.11, note that the entropy deteriorates significantly for both types of background distributions, for large k. This is because a number of embedded subspaces are not found as they blend into the background.

Performance on datasets with less dense subspaces: In this experiment we run SCHISM on Gaussian and hyper-rectangular datasets. We decrease the density of the Gaussian datasets by increasing the standard deviation of each constrained dimension in each subspace. For hyper-rectangular subspaces, each constrained dimension is constrained to an interval of width, chosen uniformly in [0.5w, 1.5w]. Thus, the density is decreased by increasing the width and hence the volume of the subspace and keeping the number of points assigned to it the same.

From Fig. 5.13 and Fig. 5.12, it is clear that as density decreases, SCHISM’s performance deteriorates. This is because a smaller percentage of the subspace’s points tend to fall into the same interval as that of the subspace center, as the volume increases. In such a case, decreasing the number of intervals in the dimension (ξ)
Figure 5.10: Effect of number of embedded subspaces \(k\) on SCHISM’s running time for different background distributions

Figure 5.11: Effect of number of embedded subspaces \(k\) on entropy of SCHISM for different background distributions

might help or we must search for less ‘interesting’ subspaces, i.e., decrease \(\tau\).

5.3.1.2 Effects of varying algorithm parameters

We now test the performance of SCHISM as its input parameters, viz. \(\tau, \xi\) are varied. This gives us an indication of the degree of difficulty in setting the free parameters. Finally, we compare performance of the SCHISM threshold function against the CLIQUE threshold function, assuming an identical test framework.
Figure 5.12: Effect of hyperrectangle subspace density \((w)\) on SCHISM’s performance

Figure 5.13: Effect of Gaussian subspace density \((w)\) on SCHISM’s performance

Effect of \(\tau\): In Fig. 5.14, we decrease the user specified interestingness threshold \(\tau\) from \(10^{-12}\) to \(10^{-25}\), and observe its effect on the coverage and running time. Note that the coverage increases rapidly, implying that \(\tau\) is the main parameter which determines how much of the search space is mined and hence running time drops rapidly too.

Effect of \(\xi\): From Fig. 5.15, we observe that, varying the number of intervals into which each dimension is discretized \((\xi)\), has a small effect on SCHISM’s perfor-
Figure 5.14: Effect of interestingness threshold (τ) on SCHISM’s performance for a considerable range of values for ξ. This is because the term ξ is incorporated into $\text{thresh}_{\text{SCHISM}}$. Outside this range however ($\xi > 15$), performance is severely degraded as the interval size becomes so small that very few contain enough points to be considered ‘interesting’.

Figure 5.15: Effect of $\xi$ on SCHISM’s performance

Effect of $\text{thresh}()$ function on performance: Here we compare the performance of the $\text{thresh}()$ function given in Eq. 3.4, with that of CLIQUE on the synthetic datasets. From Fig. 5.16, we observe that as the density of the hyper-rectangular
clusters dips due to increase in the width of its constrained dimensions, the running time of CLIQUE \(^\text{10}\) increases rapidly over that of SCHISM. Also, CLIQUE tends to split clusters into smaller ones. Its performance closely mirrors that of SCHISM for datasets having well-defined distinct clusters. However, when clusters overlap in a number of dimensions, the coverage and entropy suffers.

![Graph showing CLique vs. SCHISM running times](image)

**Figure 5.16: Effect of thresh on performance: CLIQUE vs. SCHISM**

### 5.3.2 On real datasets

**PenDigits dataset:** SCHISM outputs 128 subspaces in 4 seconds, of which the 17 clusters with the highest entropies are shown in the confusion matrix in Table 5.1. It achieves a coverage of 69\% and an entropy of 0.365 \((u = .01, \tau = 0.018)\). CLIQUE achieves a coverage of 60.7\% and an entropy of 0.49 in approximately 4 seconds too. The confusion matrix (Table 5.1), is interpreted as follows: cell \((i,j)\) denotes the number of points having true class \(i\), which were clustered by SCHISM into subspace \(j\). Ideally, each row and each column have only a single non-zero entry implying \(E(C)=0\). Note that samples of the digits \(\{3,9\}\) are both assigned by SCHISM to clusters \(C_7, C_8, C_9\) due to their similarity in structure. The clusters not shown typically have all their samples from the same digit class.

\(^{10}\)Our implementation of CLIQUE involves simply replacing \(thresh_{SCHISM}\) with \(thresh_{CLIQUE}\) in our implementation to test the significance of our threshold function
<table>
<thead>
<tr>
<th>Class</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C_4$</th>
<th>$C_5$</th>
<th>$C_6$</th>
<th>$C_7$</th>
<th>$C_8$</th>
<th>$C_9$</th>
<th>$C_{10}$</th>
<th>$C_{11}$</th>
<th>$C_{12}$</th>
<th>$C_{13}$</th>
<th>$C_{14}$</th>
<th>$C_{15}$</th>
<th>$C_{16}$</th>
<th>$C_{17}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>'0'</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>'1'</td>
<td>0</td>
<td>272</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>39</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>'2'</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>142</td>
<td>284</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>'3'</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>53</td>
<td>43</td>
<td>405</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>52</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>'4'</td>
<td>0</td>
<td>0</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>80</td>
<td>82</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>'5'</td>
<td>322</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>78</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>'6'</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>109</td>
<td>0</td>
<td>78</td>
<td>4</td>
<td>1</td>
<td>9</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>'7'</td>
<td>1</td>
<td>74</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>94</td>
</tr>
<tr>
<td>'8'</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>36</td>
<td>14</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>'9'</td>
<td>0</td>
<td>11</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>15</td>
<td>160</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>33</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.1: Confusion Matrix for PenDigits Data Set

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>FastOLGA</th>
<th>Blondel</th>
<th>Best MC Match</th>
<th>RandomMatch</th>
<th>GibbsSampling</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta$</td>
<td>0.51</td>
<td>0.51</td>
<td>0.44</td>
<td>0.97</td>
<td>0.32</td>
</tr>
<tr>
<td>$Z_{score}$</td>
<td>0.45</td>
<td>0.45</td>
<td>-3.08</td>
<td>-3.18</td>
<td>-3.15</td>
</tr>
</tbody>
</table>

Table 5.2: Performance of matching algorithms on breast cancer datasets
NBA datasets: We ran SCHISM with parameters $s = 0.075, \tau = 1/n^{-0.5} \approx 0.047, \xi = 3$. The statistics of each season yield a set of clusters/subspaces, 13 for dataset A and 11 for dataset B, for which we construct a graph using methods described in Sec. 3.2. In Table 5.3, we show members of some of the subspaces found by SCHISM from the statistics for NBA players during the 2003-04 season. The first subspace contains 37 members. These members are similar in that they all score in the bottom third in terms of number of field goals attempted, three-pointers attempted and made, free throw attempts and middle third in number of offensive rebounds. In basketball, there are primarily three positions at which the players play, viz., 'center', 'forward' and 'guard'. Within these three positions there are further variants like 'power forward', 'point guard', etc. The position at which the NBA players played, i.e., player position is not a column in our datasets. Examination of the subspace members revealed that the clusters primarily had members having the same player position. Out of the 37 members, as many as 23 (names in regular font, as opposed to italics) played in the 'center' position. In this dataset, the probability of a randomly chosen player being either 'center', 'forward' or 'guard' are given as $p('center')=0.25$, $p('forward')=0.42$, $p('guard')=0.33$. If the 37 players were drawn independently with replacement, from this distribution, the probability that $k$ of them are 'center's is binomially distributed with parameters $n = 37$ and $p = 0.25$. Accordingly, the p-value of this cluster is bounded by the area of the tail of this distribution, to the right of $k = 23$. Thus, $p-value = \sum_{k=23}^{n} {n \choose k} p^k (1-p)^{n-k} = \sum_{k=23}^{37} \left( \frac{37}{k} \right) (0.25)^k (0.75)^{37-k} = 2.16e - 05$, which may be considered to be statistically significant. As player position was not a part of the dataset, this analysis has provided us with a new insight. The second cluster has 48 members, who score in the bottom third for number of field goals and three-pointers attempted and made, and number of free throws made, middle third for average number of minutes played per game and top third for number of games played. Of the 48, 21 are 'center's. Thus, subspace mining of the NBA dataset produces statistically significant clusters w.r.t. the position the players in the cluster play at.

Breast cancer datasets: Each sample in both X and Y is labeled as either ma-
<table>
<thead>
<tr>
<th>Sample subspace members</th>
<th>Characteristic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vin Baker, Mark Blount, Chris Bosh, P.J. Brown, Marcus Camby, Tyson Chandler, Jason Collins, Samuel Dalembert, Antonio Davis, Pau Gasol, Brian Grant, Josh Howard, Marc Jackson, Antawn Jamison, Andrei Kirilenko, Raef LaFrentz, Kenyon Martin, Antonio McDyess, Stanislav Medvedenko, Yao Ming, Troy Murphy, Rasho Nesterovic, Lamar Odom, Mehmet Okur, Michael Olowokandi, Joel Przybilla, Theo Ratliff, Michael Ruffin, Bobby Simmons, Brian Skinner, Joe Smith, Stromile Swift, Etan Thomas David West, Jahidi White, Chris Wilcox, Lorenzen Wright</td>
<td>23/37 are 'center's</td>
<td>2.16e-05</td>
</tr>
<tr>
<td>Chris Andersen, Shandon Anderson, Chucky Atkins, Marcus Banks, Leandro Barbosa, Jon Barry, Tony Battie, Steve Blake, Corie Blount, Calvin Booth, Kelvin Cato, Jarron Collins, Antonio Daniels, Dale Davis, Andrew DeClercq, Boris Diaw, Howard Eisley, Reggie Evans, Jeff Foster, Dan Gadzuric, Lucious Harris, Udonis Haslem, Brendan Haywood, Jerome James, Jared Jeffries, Damon Jones, Chris Kaman, Toni Kukoc, Raúl Lopez, Mark Madsen, Walter McCarty, Chris Mihm, Nazr Mohammed, Dikembe Mutombo, Greg Ostertag, Milt Palacio, Ruben Patterson, Wesley Person, Morris Peterson, Vitaly Potapenko, Rodney Rogers, Rod Strickland, Jake Voskuhl, Charlie Ward, Earl Watson, Jiri Welsch, Aaron Williams, Jerome Williams</td>
<td>21/48 are 'center's</td>
<td>0.058</td>
</tr>
</tbody>
</table>

Table 5.3: Statistically significant subspaces from 2003-04 NBA season

Ligant or benign. Our goal is to discover these labels using unsupervised, rather than supervised learning techniques. We compare the performance of SCHISM and CLIQUE in this task. As the data is labeled, we can compute its entropy and coverage, in terms of the labels. We ran both algorithms with $u = 0.08, \xi = 3, \tau = 1/n^{-0.5} \approx 0.04$. The first line of Table 5.4 is interpreted as follows: The clustering produced by SCHISM on dataset X, has entropy 0.11 and covers 73% of X. Also, the clustering produced by CLIQUE on dataset X, has entropy 0.09 and covers
Table 5.4: Performance of SCHISM and CLIQUE on breast cancer datasets

60% of X.

Thus, CLIQUE identifies the samples it labels from dataset X a bit better than SCHISM, as it has lower entropy. However, it has significantly poorer coverage. Further examination of the subspace members, revealed that it returns 4 ‘interesting’ subspaces and primarily covers the benign samples and only 23 malignant ones out of the 414 samples it labels. For dataset Y, both algorithms have considerably worse entropy, as their coverage increases significantly. Here too, CLIQUE has worse coverage than SCHISM. For dataset Y, SCHISM and CLIQUE return 34 and 28 subspaces respectively. Thus large number of interesting subspaces typically corresponds to higher coverage and lower entropy.

Yeast microarray datasets: It is hypothesized that genes, which exhibit similar expression patterns may be co-regulated, i.e., have similar regulation mechanisms. Hence, we are looking for subspaces having similar expression patterns. We use SCHISM [71] to find these subspaces. We use $\xi = 3$. This discretizes gene expression values into three categories: under expressed (1st interval), normal (2nd interval) and over-expressed (3rd interval). Thus, the subspaces correspond to a subset of the genes/rows which are simultaneously either under-expressed or normal or over-expressed for some subset of the time samples/columns. For $s = 0.1$, SCHISM returns 4 and 9 subspaces for datasets GDS38 and GDS39 respectively.

We submitted the list of genes assigned to these subspaces to the SGD Gene Ontology(GO) Term Finder\textsuperscript{11} 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 small sample of

\textsuperscript{11}For details, see http://db.yeastgenome.org/cgi-bin/GO/goTermFinder
the results, from submitting the two of the subspaces found in dataset GDS38 by SCHISM, to the TermFinder are shown in Table 5.5. The first two rows of Table

<table>
<thead>
<tr>
<th>Gene Ontology(GO) term</th>
<th>p-value</th>
<th>Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>nucleic acid, RNA transport</td>
<td>2.49e-05</td>
<td>SUB2, NUP42, YRA1, THP2, NSP1 ...</td>
</tr>
<tr>
<td>... RNA-nucleus export</td>
<td></td>
<td>... LOS1, NUP116, NUP82, NUP188</td>
</tr>
<tr>
<td>response to temperature</td>
<td>0.00099</td>
<td>NBP2 LRE1 SPT23 HSF1</td>
</tr>
<tr>
<td>vesicle docking</td>
<td>0.00235</td>
<td>SEC1 SEC5 SEC3</td>
</tr>
<tr>
<td>regulation of growth</td>
<td>0.00349</td>
<td>NGR1 AVO1</td>
</tr>
<tr>
<td>regulation of development</td>
<td>0.00765</td>
<td>YDR306C MRK1 ATE1 VID30 AAP1 ...</td>
</tr>
<tr>
<td>protein catabolism</td>
<td>0.00496</td>
<td>... RTT101 DOA1 VPS13 SEL1 APC1</td>
</tr>
<tr>
<td>response to heat</td>
<td>0.00617</td>
<td>NBP2 LRE1 HSF1</td>
</tr>
<tr>
<td>transcription</td>
<td>0.00929</td>
<td>BRE1 PHO2 ADR1 SUM1 PTC2 ...</td>
</tr>
<tr>
<td>mitotic recombination</td>
<td>0.01735</td>
<td>... DEP1 TAF10 RAD9 GAT1 MIG1 ...</td>
</tr>
<tr>
<td>negative regulation of biological process</td>
<td>0.06285</td>
<td>... TAF1 THP2 SPT23 RGT1 MOT3 ...</td>
</tr>
<tr>
<td>... biological process</td>
<td></td>
<td>... TOP1 CFT1 RSC3 HSF1</td>
</tr>
<tr>
<td>protein de neddylation</td>
<td>0.00103</td>
<td>HO TOP1 MSH3</td>
</tr>
<tr>
<td>protein amino acid myristoylation</td>
<td>0.01168</td>
<td>BRE1 NBP2 SUM1 PTC2 ...</td>
</tr>
<tr>
<td>protein myristoylation</td>
<td>0.01168</td>
<td>... VID30 RTT101 RGT1</td>
</tr>
<tr>
<td>N-terminal protein myristoylation</td>
<td>0.01168</td>
<td>CSN9 CSII RR12</td>
</tr>
<tr>
<td>riboflavin biosynthesis, metabolism</td>
<td>0.01647</td>
<td>NMT1 FAA2</td>
</tr>
<tr>
<td>autophagy</td>
<td>0.02084</td>
<td>RIB1 RIB2</td>
</tr>
<tr>
<td>mRNA catabolism</td>
<td>0.03416</td>
<td>ATG20 ATG18 ATG7 SEC16</td>
</tr>
<tr>
<td>mitotic checkpoint</td>
<td>0.03522</td>
<td>DCS1 LSM4 DCP1 PUF2 DHH1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PPH21 ULP2 BUB2</td>
</tr>
</tbody>
</table>

Table 5.5: GO-based interpretation of genes clustered by SCHISM

5.5 are interpreted as follows: Genes SUB2, NUP42, YRA1, THP2, NSP1, LOS1, NUP116, NUP82 and NUP188 are associated with the biological processes of nucleic acid transport, RNA transport and RNA-nucleus transport. The genes belong to a subspace output by SCHISM, to which 157 genes are assigned. Out of 7274 genes in yeast, there are 70 involved in these processes. Using the right tail of the binomial distribution, GO TermFinder reports the p-value as 2.49e – 05.
5.4 Results of finding similarities across condensed models of pairs of datasets

5.4.1 On synthetic datasets

For experiments on synthetic datasets, to test the performance of algorithms to identify similarities between condensed representations of datasets, we assume that the subspace mining algorithm finds the embedded subspaces correctly, so as to isolate the contributions of those algorithms. We tested the algorithms by matching synthetic datasets having embedded subspaces. As we serially insert subspaces, for every pair of datasets, we ensure that the dataset with the larger number of embedded subspaces, includes all subspaces embedded in the other dataset. By default, we try to map a 27-vertex graph to a 34-vertex one using RandomMatch and the HIST similarity measure. For RandomMatch, we set $\delta = .03$, $g = 13$. Also, as $t$ grows very rapidly as $g$ and $|V_A|$ grow, we upper bound $t$ by 10,000. We evaluate the algorithms based on the $\#(matches)$ and its Zscore, as some parameter in generating the dataset is varied. The algorithms we compare are RandomMatch, GibbsSampling, MonteCarlo Match, FastOLGA and Blondel’s algorithm as described in [14]. The performance of FastOLGA and Blondel’s algorithm are very similar. Our experiments for thes algorithms were implemented in Matlab on a Pentium 2GHz machine running on Windows O/S with 256 MB RAM.

5.4.1.1 Interaction between type of transformation and similarity measure

In Figures 5.17 and 5.18, we observe the effect of the similarity measures HIST and AVGSIM on RandomMatch’s performance. We simultaneously vary the type of transformation between the datasets to be matched, across Noisy, Translation and Scaling. In Fig. 5.17, we plot the ratio of the Zscores of the matching produced by RandomMatch to that of the optimal matching. Typically, this ratio is positive and less than one. Ideally, it is as large as possible. From Fig 5.17, it is evident that AVGSIM is outperformed by HIST for the Noisy and Translation transformation, while for Scaling, it is inconclusive. Further, HIST often outperforms optimal as RandomMatch reports matchings smaller than the optimal. Also, in Fig. 5.18,
neither algorithm has a significant value for \( \#(\text{matches}) \). Hence, we choose HIST for the remainder of the experiments. Also, this shows that our algorithms are relatively resilient to a number of transformations, due to the high ratio of Zscores.

![Graph](image)

**Figure 5.17:** Ratio of Zscores v/s type of dataset transformation

![Graph](image)

**Figure 5.18:** \( \#(\text{matches}) \) v/s type of dataset transformation
5.4.1.2 Effect of varying dataset parameters on $\#(\text{matches})$

In Figures 5.22, 5.23 and 5.24, we compare the performance of the mapping algorithms in terms of $\#(\text{matches})$ as some parameter, viz., $c, o$ and $p$, used in generating the embedded subspaces is varied. Note that no algorithm stands out for $\#(\text{matches})$. Also, note that GibbsSampling is highly susceptible to perturbation, whereas Blondel’s algorithm is relatively resilient, as seen from Fig. 5.24.

5.4.1.3 Effect of varying dataset parameters on Zscore

In Figures 5.19, 5.20 and 5.21, we compare the effect of the different matching algorithms on the Zscore. The clear winner is RandomMatch, which occasionally produces a lower Zscore than the optimal matching as well. This is because the optimal matching typically involves more matched pairs of vertices than that produced by RandomMatch. The extra pairs of matches raise the Zscore of the optimal matching.

GibbsSampling is typically outperformed by all the other algorithms for the Zscore measure. This occurs because it samples only 100 matchings from the matching space which is very large and tries to optimize on these matches. Thus, it samples a much smaller space than the other space search algorithms like RandomMatch and MonteCarlo Match, which typically sample about 10,000 matchings from the matching space. Using 10,000 iterations for GibbsSampling takes unacceptably long. This also explains its good performance for $k = 10$ in Fig. 5.25.

Another interesting observation is the Zscore curve for the optimal matching. For $c = 0.1$ in Fig. 5.19, it is greater than that for the average matching obtained via MonteCarlo sampling. This implies that for highly-constrained subspaces, i.e., $c$ is small, HIST may not perform too well at expressing similarities between subspaces or Zscore may not be a reliable indicator. Alternatively, it may be the case that, for small $c$, matchings are highly susceptible to perturbation. As $c$ increases, Zscore for the optimal matching decreases considerably and there exists a strong correlation between goodness of match and Zscore. Also, for small $k$ as in Fig. 5.25, it is relatively large. This indicates that if a few subspaces are embedded, there exist matchings, different from the optimal one, which produce Zscores, which are not
From Fig. 5.21, it is evident that at higher $p$-values, i.e., $p \geq 0.1$, discrimination between the optimal and MonteCarlo matchings, using $Z_{score}$ is mitigated as the optimal matching is less than eight standard deviations from the mean MC matching, which relative to typical optimal matching $Z_{score}$ values, as seen in the graphs in Sec. 5.4.1, is not too large.

![Graph 1](image1)

**Figure 5.19:** $Z_{score}$ v/s $c$, subspace dimensionality parameter

![Graph 2](image2)

**Figure 5.20:** $Z_{score}$ v/s $o$, overlap parameter
5.4.1.4 Effect of varying dataset parameters on \( #(matches) \)

In Figures 5.22, 5.23 and 5.24, we compare the performance of the mapping algorithms in terms of \( #(matches) \) as some parameter, viz., \( c, o \) and \( p \), used in generating the embedded subspaces is varied. Note that no algorithm stands out for \( #(matches) \). Also, note that GibbsSampling is highly susceptible to perturbation, whereas Blondel’s algorithm is relatively resilient, as seen from Fig. 5.24.
Figure 5.23: \#(matches) v/s \(o\), overlap parameter

Figure 5.24: \#(matches) v/s \(p\), amount of perturbation

5.4.1.5 Scalability of algorithms

In Figures 5.25, 5.26, and 5.27 we compare the performance of RandomMatch, MonteCarlo Match and GibbsSampling with that of Blondel’s algorithm [14]. Here too, RandomMatch, is clearly the best performer in terms of Zscore. On an average, it is also the best performer for \#(matches). On an average, Blondel’s algorithm takes far less time than the others as is visible from Fig. 5.27. However, as we limit the number of trials in RandomMatch to 10,000, it
completes faster than all the other methods for \( k = 75 \), while still producing the best \( Z \text{score} \). **GibbsSampling** is computationally, the most expensive, despite its poor \( Z \text{score} \).

![Graph showing Zscore vs k](image)

**Figure 5.25: Zscore v/s k, number of embedded subspaces**

![Graph showing #matches vs k](image)

**Figure 5.26: \#(matches) v/s k, number of embedded subspaces**

All the algorithms have running time independent of \( n \) and \( d \). Hence, results for these are not shown.
5.4.2 On real datasets

**NBA datasets:** As stated in the introduction to this paper, our framework may be applied to study evolution of datasets over time. For instance, we may wish to answer the question: What set of clusters have *not* changed over time? For the NBA datasets over two seasons, this may translate as: which groups of players have similar performance across the two seasons? If models of performance for the two seasons yield structurally similar clusters which overlap in their members, then these overlapping members are likely to have very similar performance. Consider the following scenario in which such knowledge may be employed: let players, say E, F and G, currently belonging to distinct teams P, Q and R respectively, all wish to leave their current team. If by our clustering models for the two seasons, it is known that players, E and F, belong to structurally similar clusters, then they show similar performance across the two seasons, prompting management at P and Q to consider “exchanging” them.

We structurally matched the two graphs, obtained using **SCHISM**, as described in Sec. 5.3.2 and **HIST**, using **OLGA**, described in Sec. 4.2. For each pair of matched clusters, we report the intersection set of players. We found two clusters, having more than two members each, as shown in Figure 5.6, preserved structurally, with respect to the rest of the dataset, across the two years. The first subspace
mined from the 2003-2004 NBA season dataset, as shown in Table 5.3 is matched
with a subspace from the 2004-05 NBA season dataset, having 84 members, which
score in the lower third for number of field goals three-pointers and free-throws at-
ttempted, number of turnovers and middle third for the number of assists. Thus,
these two clusters are similar in that they are both in the lower third for number
of field goals, three-pointers and free-throws attempted. The intersection of the
names of this pair of matched subspaces has 24 members. Across both datasets, the
probabilities of a randomly chosen player being either ‘center’, ‘forward’ or ‘guard’
are approximately the same and are given as \( p(\text{‘center’}) = 0.25 \), \( p(\text{‘forward’}) = 0.42 \),
\( p(\text{‘guard’}) = 0.33 \). If the 24 players were drawn independently with replacement,
from this distribution, the probability that \( k \) of them are ‘center’s, is binomially
distributed with parameters \( n = 24 \) and \( p = 0.25 \). Accordingly, the p-value of this
cluster is bounded by the area of the tail of this distribution, to the right of \( k = 16 \).
Thus, \( \text{p-value} = \sum_{k=16}^{n} \binom{n}{k} p^k (1-p)^{n-k} = \sum_{k=16}^{24} 4 \binom{24}{k} (0.25)^k (0.75)^{24-k} = 1.37e-04 \),
which may be considered to be statistically significant. As player position was not
a part of the dataset, this analysis has provided us with a new insight. Further-
more, in this cluster, of the eight members not listed as ‘center’s, four, viz., Chris
Bosh, Kenyon Martin, Antonio McDyess and Joe Smith, play in the position of
power-forward, which is the most similar position to ‘center’.

The second cluster has 25 players, of which 18 are ‘guard’s. Thus, the p-value
is 0.0023. These 25 players are part of a cluster mined from the dataset for 2003-04
NBA season which has 73 members of which 36 are ‘guard’s, having a p-value of
0.055. Thus, structurally matching the clusters, yields a higher p-value, i.e., a purer
cluster.

**Breast cancer datasets:** Another possible application of our framework is
the mining of related but schematically differing datasets. After creating the graphs
and matching the clusters structurally, we examine the labels of the samples in
matched clusters. We ran **RandomMatch** on the subspaces output by **SCHISM**
as mentioned in Sec. 5.3.2, with parameters \( g = 4 \), \( \delta = 0.03 \),

Let \( p(B, u \in V_X) \), \( p(B, v \in V_Y) \) denote the probability that a sample drawn
uniformly at random from clusters \( u, v \) respectively, from graphs corresponding to
<table>
<thead>
<tr>
<th>Common cluster members</th>
<th>Characteristic</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark Blount, Chris Bosh, Jason Collins, Samuel Dalembert, Antonio Davis, Josh Howard,</td>
<td>16/24 are</td>
<td>1.37e-04</td>
</tr>
<tr>
<td>Marc Jackson, Andrei Kirilenko, Raef LaFrentz, Kenyon Martin, Antonio McDyess, Yao Ming,</td>
<td>'center's</td>
<td></td>
</tr>
<tr>
<td>Rasho Nesterovic, Mehmet Okur, Michael Olowokandi, Joel Przybilla, Theo Ratliff, Bobby Simmons, Brian Skinner, Joe Smith, Stromile Swift, Etan Thomas, Lorenzen Wright, Pau Gasol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rafer Alston, Chauncey Billups, Speedy Claxton, Ricky Davis, Manu Ginobili, Drew Gooden,</td>
<td>18/25 are</td>
<td>0.0023</td>
</tr>
<tr>
<td>Richard Hamilton, Jarvis Hayes, Kirk Hinrich, Jim Jackson, Marko Jaric, Desmond Mason,</td>
<td>'guard's</td>
<td></td>
</tr>
<tr>
<td>Jeff McInnis, Andre Miller, Brad Miller, Mike Miller, Cuttino Mobley, Steve Nash, Tony Parker, Gary Payton, Damon Stoudamire, Jason Terry, Kurt Thomas, Jamaal Tinsley, David Wesley</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.6: Structurally similar clusters from two NBA seasons

<table>
<thead>
<tr>
<th>((u \in V_X, v \in V_Y))</th>
<th>(0,11)</th>
<th>(1.27)</th>
<th>(2,12)</th>
<th>(3,28)</th>
<th>(4,20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p(B, u))</td>
<td>0.955</td>
<td>0.984</td>
<td>0.888</td>
<td>0.714</td>
<td>0.012</td>
</tr>
<tr>
<td>(p(B, v))</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.625</td>
<td>0.0</td>
</tr>
<tr>
<td>((p(B, u) - p(B, v))^2)</td>
<td>0.002</td>
<td>0.0001</td>
<td>0.0123</td>
<td>0.0079</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

Table 5.7: Structurally similar clusters from schematically different breast cancer datasets

datasets X, Y respectively, is labeled ‘benign’. Then if our framework finds that \(P_f(u, v) = 1\), from Eq. 1.1, i.e., the cluster \(u\) of \(X\), is matched to cluster \(j\) of \(Y\), we found that \(p(B, u) \approx p(B, v)\), i.e., we found a strong correlation between labels of elements of matched clusters. In Table 5.7, we report the probabilities of \(p(B, u \in V_X)\) and \(p(B, v \in V_Y)\), \(\forall P_f(u, v) = 1\). The first column is interpreted as cluster 0 of the first dataset (X) has all but 4 of its 89 elements labeled as ‘benign’, i.e., \(p(B, u = 0) = 1 - 4/89 = 0.955\), while cluster 11 of the second dataset (y) has both its elements labeled as ‘benign’, i.e., \(p(B, v = 11) = 2/2 = 1.0\). Such findings allow us to search for correlations between the two spaces corresponding to X and
Y. We measure this correlation between the matched clusters using

$$\beta(f) = \exp(-\sum_{v \in V_Y} (p(B, v) - p(B, f(v)))^2)$$

Note that $\beta()$ converts the Euclidean distance between the probabilities of a sample being labeled ‘benign’ in the matched subspaces from the two datasets, into a similarity measure in the range [0,1].

Although, the clusters found in both datasets are predominantly benign, our algorithm correctly matches the benign ones, i.e., clusters 0 and 1 of X with clusters 11 and 27 of Y respectively, the malignant one, i.e., cluster 4 of X with cluster 20 of Y, as well as the higher entropy clusters, i.e., 3 of X with 28 of Y.

If X and Y were datasets collected from different hospitals, which could not directly exchange the datasets, but could exchange $G_X$ and $G_Y$, then as $p(B, u) \approx p(B, v)$, the treatments which are successful on patients assigned to $u$, may be considered for patients assigned to $v$, without ever actually exchanging X and Y directly. Further, even if X and Y were collected from the same hospital and sharing were allowed, the difference in their schema poses serious problems in learning from both of them. Accordingly, the characteristics of the matched subspaces could be examined for correlations. Thus, correlation of distribution within matched components, where the matching is based on structural similarity between schematically different datasets, can help provide many insights. From Table 5.2, it is clear that RandomMatch outperforms the other algorithms both in terms of Zscore and $\beta()$. Also, GibbsSampling and MC Match perform well on Zscore as the size of matching is small. Finally note that, FastOLGA and Blondel’s algorithm perform well on $\beta()$, despite a high Zscore.

Yeast cell cycle microarray datasets: SCHISM returns 4 and 9 subspaces for datasets GDS38 and GDS39 respectively. A sample of the analysis of these subspaces, using the Gene Ontology TermFinder is shown in Sec. 5.3.2. We then construct the graphs for each dataset and structurally match the underlying subspaces/vertices using RandomMatch. 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\textsuperscript{12} 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 small sample of their results are shown in Table 5.8.

<table>
<thead>
<tr>
<th>Gene Ontology (GO) term</th>
<th>p-value</th>
<th>Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>chromatin silencing at telomere</td>
<td>0.00068</td>
<td>BRE1, SUM1</td>
</tr>
<tr>
<td>telomeric heterochromatin formation</td>
<td>0.00068</td>
<td>BRE1, ADR1, SUM1</td>
</tr>
<tr>
<td>gene, chromatin silencing</td>
<td>0.00215</td>
<td></td>
</tr>
<tr>
<td>regulation of metabolism</td>
<td>0.00561</td>
<td></td>
</tr>
<tr>
<td>transcription</td>
<td>0.00855</td>
<td></td>
</tr>
<tr>
<td>organelle organization and biogenesis</td>
<td>0.00918</td>
<td>BRE1, ADR1, SUM1, SPC110</td>
</tr>
<tr>
<td>ribosome biogenesis</td>
<td>4.13e-05</td>
<td>MAK16, SPB4, CGR1 ...</td>
</tr>
<tr>
<td>ribosome biogenesis and assembly</td>
<td>0.0001</td>
<td>... TSR2, RLP7, NOP4 ...</td>
</tr>
<tr>
<td>dicarboxylic acid transporter activity</td>
<td>0.00059</td>
<td>SFC1, DIC1</td>
</tr>
<tr>
<td>recombinase activity</td>
<td>0.00059</td>
<td>KEM1, RAD52</td>
</tr>
<tr>
<td>cytoskeletal protein binding</td>
<td>0.00251</td>
<td>NUM1, BNR1, ASE1, MLC2</td>
</tr>
<tr>
<td>transcription cofactor activity</td>
<td>0.00885</td>
<td>SPT8, SWI6, ARG81</td>
</tr>
<tr>
<td>signal transduction</td>
<td>0.01704</td>
<td>COS111, BEM2</td>
</tr>
<tr>
<td>DNA replication</td>
<td>0.00194</td>
<td>ECO1, DPB2</td>
</tr>
<tr>
<td>DNA repair</td>
<td>0.00402</td>
<td></td>
</tr>
<tr>
<td>response to DNA damage stimulus</td>
<td>0.00551</td>
<td></td>
</tr>
<tr>
<td>response to endogenous stimulus growth</td>
<td>0.00551</td>
<td></td>
</tr>
<tr>
<td>growth</td>
<td>0.0078</td>
<td>TEC1, BEM2</td>
</tr>
<tr>
<td>electron transport</td>
<td>1.24e-05</td>
<td>RIP1, COX4, COX6</td>
</tr>
<tr>
<td>oxidative phosphorylation</td>
<td>4.01e-05</td>
<td></td>
</tr>
<tr>
<td>aerobic respiration</td>
<td>0.00022</td>
<td></td>
</tr>
<tr>
<td>intracellular signaling cascade</td>
<td>0.00213</td>
<td>SAC7, BOI2</td>
</tr>
<tr>
<td>signal transduction</td>
<td>0.00503</td>
<td></td>
</tr>
<tr>
<td>protein biosynthesis</td>
<td>0.01050</td>
<td>SUP45, SES1, YDR341C</td>
</tr>
<tr>
<td>lipid biosynthesis</td>
<td>0.00267</td>
<td>SUR4, KES1</td>
</tr>
<tr>
<td>cellular lipid metabolism</td>
<td>0.00722</td>
<td></td>
</tr>
<tr>
<td>lipid metabolism</td>
<td>0.00823</td>
<td></td>
</tr>
<tr>
<td>development</td>
<td>0.01134</td>
<td>BRE1, ERV14</td>
</tr>
<tr>
<td>organelle organization and biogenesis</td>
<td>0.05258</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.8: GO-based interpretation of similar substructure

The first row of Table 5.8 is interpreted as follows: Genes SUM1 and BRE1 are associated with the process of chromatin silencing at telomere. These genes actually

\textsuperscript{12}For details, see http://db.yeastgenome.org/cgi-bin/GO/goTermFinder
belong to a cluster of 7 genes but out of 7274 genes in yeast there are 42 involved in this process. Using the right tail of the binomial distribution, GO TermFinder reports the p-value (measure of statistical significance) as 0.00068. Further, they are also associated with gene silencing and the p-value is 0.00215. SUM1 and BRE1 belong to a subspace of 193 genes when SCHISM is applied to dataset GDS38. This results in a much lower p-value and are hence not reported as statistically significant. This is true for a number of other clusters reported too. Thus, the condensed model approach yields smaller, more statistically interesting clusters, by leveraging information from multiple sources.

Furthermore, by varying the input parameters \( \tau \) to SCHISM, we can vary the granularity of the subspaces discovered, thereby generating fine-grained or coarse models of a single dataset. This provides us with different models having slight heterogeneity for the same original dataset. The fine-grained model may not cover all the genes covered in the coarser model. Hence, the set of biological processes, with which the genes in the clusters of the fine-grained model are involved, may differ from those of the coarser model. Thus, by applying RandomMatch to the different models, we get a different matching, which also provides biologically meaningful and statistically significant clusters. In one such coarser new model, BRE1 is clustered with ERV14, as visible at the bottom of Table 5.8, as opposed to SUM1 (not covered by the new model) at the top of the table.
CHAPTER 6
CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

From our experiments in Sec 5.3.1 and Sec. 5.3.2, it is evident, that we succeed in finding components in high-dimensional datasets. We have tested our algorithms on real datasets having as many as 30 dimensions \[12\] and synthetic datasets having as many as 300 dimensions (see Fig. 5.5). \textbf{SCHISM} reports low entropy values and reasonably high coverage (see Sec. 5.1.1) for these datasets. Furthermore, we tested the proposed similarity measures HIST and AVGSIM, in these experiments and the former seems to have better performance on an average. Also \textbf{SCHISM} outperforms \textbf{CLIQUE} on both; the breast cancer \[89\] and the PenDigits \[2\] datasets \[13\].

In Sec. 6.2.1.2, we provide a formulation of the problem of aligning multiple graphs used to represent datasets. We show that the formulation is a special case of the Minimum Multicut problem, which is known to have to have an approximation algorithm \[34\], which has approximation ratio \(O(log(|V|))\), where \(|V|\) is the number of vertices in the underlying graph having maximum number of vertices.

In Sec. 5.4.1 and Sec. 5.4.2, we test the matching algorithms proposed in Chapter 4. The algorithm \textbf{RandomMatch} outperforms the other ones proposed and that of Blondel \[14\] in terms of \textit{Zscore} as defined in Sec. 5.1.2. This algorithm provides additional advantages. For example, it can return multiple non-disjoint matchings. Also, it can return matchings between subgraphs of the condensed representations of the pairs of datasets.

6.2 Future Work

Open problems that we plan to address in the future include

1. Design of new similarity measures: We have shown application of our methods

\[13\]The NBA and microarray datasets are unlabeled and hence entropy cannot be evaluated
to microarray data in Sec. 5.4.2. Currently, computational biologists are searching for far more complex patterns than axis-parallel hyperrectangles, viz. coherent patterns, etc. To generate condensed representations of datasets using such complex patterns motivates the design of new similarity measures, other than HIST and AVGSIM.

2. Extension to multiple datasets: Typically, there exist multiple sources of related data. Our problem is NP-hard for two datasets itself. Hence, extending our algorithms to multiple datasets seems a formidable, yet essential task.

3. Indexing of condensed representations of datasets: by indexing condensed representations of datasets, we can submit a query condensed representation and find similar datasets. We hypothesize that this may expose similarities in datasets across different domains and highlight availability of solutions and analyses from other disciplines. This could develop into a way to automate interdisciplinary research and analysis.

4. Extension to higher-order tensors: A first-order tensor is a vector, a second-order tensor is a matrix. There exist ($n \geq 3$)-order tensors as well. This poses questions like: how do we generate condensed representation for such higher-order tensors? How do we produce matchings across them?

6.2.1 Extending the solution to multiple graphs

We propose two possible solutions to the problem of finding similarities across condensed representations of multiple datasets. We first try to extend FastOLGA to multiple datasets. Our second attempt reduces the problem to a special case of a much-studied problem, i.e. the Minimum Multicut problem [76], for which an approximation algorithm is known.

6.2.1.1 Extending FastOLGA

In Sec. 4.2, we recursively defined similarity, by using the concept “Two objects are similar, if objects they are related to are similar”. Formalization of this
definition of similarity yields a power-method based algorithm, FastOLGA, as described in Fig. 4.3. To find similarities between the models for the two datasets, FastOLGA attempts to align the dominant eigenvectors of the models of the two datasets. To align multiple datasets, would then intuitively suggest an algorithm, in which we align the dominant eigenvectors of each of the graphs of the datasets. However, there exist some limitations to our approach, which stem from the limitations of OLGA. Hence, we propose to incorporate the following features into future algorithms: Firstly, as in Sec. 4.4, we consider the possibility that there may be structurally similar subgraphs appearing in only a subset of the models for the datasets under consideration. Secondly, there may be a set of structural subgraphs appearing over possibly overlapping subsets of the models of datasets under consideration. Hence, it appears more prudent to align the dominant eigenvectors of subsets of the models of the datasets under consideration, rather than all the dominant eigenvectors.

6.2.1.2 Extending the solution to multiple graphs

Consider the case where we have multiple graphs, i.e., we are given dataset \( DB = \{G_i(V_i, E_i)|i \in [1, k]\} \) of \( k \) complete graphs, and our goal is to find an optimal matching across these graphs. This is difficult, as it is known that tripartite matching, and in general for \( k > 2 \), that the \( k \)-partite weighted maximal matching problem is NP-complete [65]. One avenue worth exploring, is the utilization of our work in Chapter 4 to find similarities between all pairs of vertices across all pairs of the graphs in \( DB \). We can then combine the graphs into a single graph \( G = (V = \cup V_i, E \subset \{(u, v) : u \in V_i, v \in V \setminus V_i, i \in [1, k]\}, SIM) \), where \( SIM \) contains the similarity matrices between all pairs of graphs in \( DB \) at appropriate places. For example, the set of similarity matrices between all pairs of graphs may be denoted as \( Sim = \{\Pi_{G_i,G_j} : i \in [1, k], j \in [1, k]\} \). Then

\[
SIM = \begin{pmatrix}
\text{zeros}(|V_1|, |V_1|) & \Pi_{G_1,G_2} & \cdots & \Pi_{G_1,G_k} \\
\Pi_{G_2,G_1} & \text{zeros}(|V_2|, |V_2|) & \cdots & \Pi_{G_2,G_k} \\
\vdots & \vdots & \ddots & \vdots \\
\Pi_{G_k,G_1} & \Pi_{G_k,G_2} & \cdots & \text{zeros}(|V_k|, |V_k|)
\end{pmatrix}
\]

The problem of finding an optimal matching may then be posed as the question:
“Which set $F \subseteq E$ of edges, must be removed from the graph $G$, having minimum sum of weights, while ensuring that each resulting connected component contains vertices from distinct graphs in DB?”

**The Minimum Multicut problem** This problem [76], is posed as follows: 

*Given an undirected graph $G = (V, E)$, $t$ pairs of vertices $(s_i, t_i), \ i = 1 \ldots t$ and costs $c_e \geq 0$ for each edge $e \in E$, find a minimum-cost set of edges $F$ such that for all $i$, $s_i$ and $t_i$ are in different connected components of $G' = (V, E - F)$. 

As we wish to separate all pairs of vertices within the same underlying graph in DB, the k-partite weighted maximal matching problem is a special case of the Minimum Multicut problem and $t = \sum_{i=1}^{k} (|V_i|)$. Garg et al. [34], have presented an algorithm GVY for the minimum multicut problem and we use the same to solve our problem. We first define indicator variables for each edge in $E$, such that

\[ \forall e \in E, x_e = \begin{cases} 
1 & \text{if } e \in F \\
0 & \text{otherwise} 
\end{cases} \quad (6.1) \]

Then, the sum of weights on any path in $G$ to be broken, due to the removal of edges in $F$ exceeds 1. Accordingly, this problem can be formulated as an integer programming problem and written as:

\[
IP = \min \sum_{e \in E} c_e x_e
\]

subject to:

\[
\sum_{e \in P} x_e \geq 1 \quad \forall P \in P_{uv}, \ \forall u \in V_i, \ v \in V_i \setminus u, \ i \in [1, k]
\]

\[
x_e \in \{0, 1\}
\]

where $P_{uv}$ is the set of all paths between vertices $u, v$ within graphs in DB. Relaxing
it to the linear program, we have:

\[ LP = \min \sum_{e \in E} c_e x_e \]

subject to:

\[ \sum_{e \in P} x_e \geq 1 \quad \forall P \in P_{uv}, \forall u \in V_i, v \in V_i \setminus u, i \in [1, k] \]

\[ x_e \geq 0 \]

We then use the GVVY algorithm as shown in Fig. 6.1. As the number of constraints

<table>
<thead>
<tr>
<th>GVVY ((G, DB))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( F = \emptyset )</td>
</tr>
<tr>
<td>2. Solve LP and get optimal solution ( x = { x_e : e \in E } )</td>
</tr>
<tr>
<td>3. while ( \exists ) some ( (u \in V_i, v \in V_i, i \in [1, k]) ) such that ( \max_{P_{uv}} \sum_{e \in P} x_e &lt; 1 )</td>
</tr>
<tr>
<td>4. Let ( S = { a : \min_{P_{uv}} \sum_{e \in P} x_e \leq r &lt; 0.5 } )</td>
</tr>
<tr>
<td>5. ( \delta(S) = { (l, m) : l \in S, m \notin S } )</td>
</tr>
<tr>
<td>6. ( F = F \cup \delta(S) )</td>
</tr>
<tr>
<td>7. Remove ( S, \delta(S) ) from current graph</td>
</tr>
<tr>
<td>8. return ( F )</td>
</tr>
</tbody>
</table>

**Figure 6.1: GVVY algorithm for minimum multicut**

is possibly exponential, they suggest use of an ellipsoid algorithm to solve the linear program. The algorithm separates at least 1 pair of vertices \((u, v)\) in each iteration (line 3), by constructing a ball \( S \) of radius \( r \) around \( u \) (line 4). They find all edges \( (l, m) \), such that the sum of the similarities on any path from \( u \) to \( m \) exceeds \( r \) and the minimum sum of the similarities over all paths from \( u \) to \( l \) is not exceeded by \( r \). All such edges having one vertex lying inside and the other outside of \( S \), i.e., \( \delta(S) \), are removed from \( G \) and added to \( F \) (line 6). Also all edges and vertices in \( S \) are removed from \( G \) (line 7). This is a \( 4ln(2t) \)-approximate algorithm.
BIBLIOGRAPHY


[94] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In IEEE International Conference on Data Mining, Maebashi City, 2002.