Technische Universität München
Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme

Analyzing and Predicting Large Vector-, Graph- and Spatio-Temporal Data

Nina Christine Hubig

Technische Universität München
Fakultät für Informatik
Lehrstuhl III – Datenbanksysteme

Analyzing and Predicting Large Vector-, Graph- and Spatio-Temporal Data

Diplom-Informatikerin Univ.
Nina Christine Hubig

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Martin Bichler

Prüfer der Dissertation: 1. Prof. Alfons Kemper, Ph.D
2. Hon.-Prof. Dr. Thomas Runkler

Die Dissertation wurde am 16.05.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 29.08.2017 angenommen.

For everyone,
who supported me.
Thank you. :)

Abstract

Large social graph datasets, pertaining to millions of social network users and the billions of relationships between them; complex, high dimensional vector data of large database systems; and petabytes of environmental sensor data are being generated every day. Employing this flood of data for the benefit of all, is one of the main challenges of the 21st century[129, 88, 51].

This thesis advances the field of data mining and machine learning for a variety of data types. For vector data two novel *subspace clustering* techniques are introduced, focusing on redundancy reduction and automation to increase the efficiency of the algorithms. For graph data, this thesis proposes two novel clustering methods, i.e., *community detection* methods, based on minimal patterns with again no-redundancy and automation. Automation of all algorithms is achieved by creating a coding scheme using the minimum description length principle[161]. Further, an efficient *classification* method, predicting natural hazards, specifically tropical storms, using extremely large environmental sensor data from historic timeseries is developed. High efficiency is achieved by applying *tensor factorization* techniques to large graph and timeseries data. Last, this thesis shows a way to seamlessly integrate and automatically optimize these methods in *modern relational main-memory databases*, employing examples of classical clustering and classification approaches.

All introduced methods are vastly experimentally evaluated and have already contributed to the respective research society in most cases. The integration of these methods into relational main-memory databases allows a wide leap from the theoretical method creation process to praxis-oriented database usage.

Kurzfassung

Große soziale Graph-datensätze, die zu Millionen von Nutzern sozialer Netzwerke und den Milliarden Beziehungen zwischen ihnen gehören; komplexe, hochdimensionale Vektordaten von großen Datenbanksystemen; und Petabytes von Umweltsensordaten, die jeden Tag generiert werden. Diese Flut an Daten nutzbar zu machen zum Gewinn aller, ist eine der größten und wichtigsten Herausforderungen des 21. Jahrhunderts [129, 88, 51].

Diese Arbeit bringt das wissenschaftliche Feld von Data Mining und Maschinellem Lernen für eine Auswahl solcher Datentypen voran: Für Vektordaten werden in dieser Arbeit zwei *Subspace Clustering* Techniken vorgestellt, die sich um die Effizienz dieser Verfahren zu steigern auf Redundanzreduzierung und Automatisierung fokussieren. Für Graphdaten stellt diese Arbeit zwei neue Clusteringmethoden bzw. das *Herausbilden von Gruppen in Graphen* vor, die zum einen auf der Erkennung minimaler Muster innerhalb solcher Gruppen basieren und auch wiederum Redundanzreduktion und Automatisierung beinhalten. Die Automatisierung aller Algorithmen wird durch die Generierung eines Kodierungsschematas der minimalen Beschreibungslänge (einer Kompressionsmethode) [161] erreicht. Des weiteren betont diese Arbeit eine effiziente *Klassifizierungsmethode*, welche Naturkatastrophen, insbesondere tropische Stürme, auf großen Umweltsensordaten historischer Zeitreihen vorhersagt. Hohe Effizienz der Algorithmen auf großen Graph- und Zeitreihendaten wird durch das Anwenden der Tensorfaktorisierungs-Technik erreicht. Zuletzt zeigt diese Arbeit wie man Methoden dieser Art nahtlos und automatisch optimiert in moderne relationale *Hauptspeicherdatenbanken* integrieren kann. Dies wird am Beispiel klassischer Clustering- und Klassifikationsverfahren verdeutlicht.

Alle eingeführten Methoden sind weitreichend experimentell evaluiert und haben in vielen Fällen bereits einen Beitrag in der jeweiligen Wissenschaft geleistet. Die Integration von all diesen Methoden in relationale Hauptspeicherdatenbanken zeigt einen weiten Sprung zwischen dem theoretischen Methodenkreierungsprozess und der praxisorientierten Datenbanknutzung.

Contents

Abstract vii

Kurzfassung ix

List of Figures xv

List of Tables xxi

1 Introduction 1
- 1.1 Knowledge Discovery in Heterogeneous Data 3
  - 1.1.1 Vector Data 4
  - 1.1.2 Spatio-Temporal Data 5
  - 1.1.3 Graph Data 5
- 1.2 Important Techniques and Systems 7
  - 1.2.1 Subspace Clustering 7
  - 1.2.2 Outlier Detection 8
  - 1.2.3 Classification 9
  - 1.2.4 Minimum Description Length 10
  - 1.2.5 Tensor Factorization 11
  - 1.2.6 HyPer 12
- 1.3 Contribution and Outline 15

2 Vector Data 17
- 2.1 NORD - Non-redundant Subspace Clustering 17
  - 2.1.1 Goal: Balancing Quality and Novelty 20
  - 2.1.2 Algorithm: NORD 28
  - 2.1.3 Experimental Evaluation 32

2.1.4 Related Work 39
2.2 ISAAC - ISA Arbitrary (Subspace) Clustering 41
2.3 ISAAC - ISA Arbitrary Subspace Clustering 42
2.3.1 Contributions 44
2.3.2 Preliminaries 46
2.3.3 Generalized Independent Subspace Clustering 50
2.3.4 Algorithm 58
2.3.5 Experimental Evaluation 62
2.3.6 Related Work And Discussion 72
2.4 Conclusion 75

3 Graph Data 77

3.1 IROC - Non-redundant Overlapping Clustering 77
3.1.1 Problem Definition 80
3.1.2 Coding Scheme 84
3.1.3 Algorithm: IROC 90
3.1.4 IROC meets Tensor Factorization 96
3.1.5 Experimental Evaluation 98
3.1.6 Related Work and Discussion 107
3.2 CXprime - Automated Clustering Using Structure Primitives 109
3.2.1 Graph Compression 112
3.2.2 Algorithm: CXprime 119
3.2.3 Experimental Evaluation 123
3.2.4 Link Prediction 133
3.3 Conclusion 135

4 Spatio-temporal Data 137

4.1 Introduction 138
4.2 Spatio-Temporal Outlier Detection 141
4.2.1 Algorithmic procedure 145
4.3 Framework for Natural Hazards Detection 146
4.3.1 Environmental Tensor Factorization 147
4.3.2 Classifying Natural Hazards 150
4.4 Experimental Evaluation 150
4.4.1 Global Climate Data 151
4.4.2 Finding synthetic spatio-temporal outliers 152

4.4.3 Finding Natural Hazards on Real Data 154
4.5 Related Work 156
4.6 Conclusion 157

5 Data Mining in Main Memory DB Systems 159
5.1 Introduction 160
5.1.1 Contributions 163
5.1.2 Structure 164
5.2 Related Work 164
5.2.1 Dedicated Data Analytics Tools 164
5.2.2 Data Analytics in Databases 166
5.3 HyPer For Data Analytics 167
5.4 In-database Processing 169
5.4.1 Program Execution within the Database 169
5.4.2 Extensions to SQL 170
5.4.3 Data Analytics in the Database Core 170
5.5 Data Analytics using SQL 172
5.5.1 The Iterate Operator 173
5.5.2 Query Optimization and Seamless Integration with the Surrounding SQL Query 175
5.6 Operators 176
5.6.1 The Physical k-Means Operator 177
5.6.2 The Physical Naive Bayes Operator 178
5.6.3 The Physical PageRank Operator 179
5.7 Lambda Expressions 180
5.8 Experimental Evaluation 181
5.8.1 Datasets and Parameters 182
5.8.2 Evaluated Systems 185
5.8.3 Evaluation Machine 185
5.8.4 Results and Discussion 186
5.9 Conclusion 188

6 Conclusion 191

Bibliography 195

List of Figures

1.1 The knowldege discovery in databases process. 2

1.2 Vector Data: numerical (age, wage) and categorical (gender) attributes of a relation. 4

1.3 (a) axis parallel clustering shapes (b) arbitrary clustering shapes . . 7

1.4 Minimum Description Length: Compressing the same data with three different models. The shortest bitstring in case (2) of model (grey) and data (white) yields the best representation. (Image Source [156]) 10

1.5 PARAFAC decomposition of a three-way tensor as a sum of n outer products (rank-one tensors). 11

2.1 A possible redundant output of an unspecific subspace clustering method depicted with our new visualization technique. The clusters C_1 to C_5 show their redundancy due to their position on the information plane. The higher the distance d between the clusters, the less redundant they are to each other. 18

2.2 The information content of two cluster. The overlapping part of cluster 1 and cluster 2 is redundant, the other parts are novel information provided by this one cluster. Together, novelty and redundancy form the full information of a cluster. 22

2.3 Figure Left: Sending order of the clusters of a clustering to a receiver. While the first cluster is sent in total, all subsequent ones are only sending their novel information. Figure Right: Estimating an arbitrary oriented cluster by microclusters with $m = 5$. 25

2.4 Visualization of quality and novelty on a synthetic example data set. Separated for the objects and the subspaces of the clustering. While the clusters seem correlated in the subspaces, their objects differ a lot. Best viewed in color. 28
2.5 Quality and runtime evaluation of inner parameter m. 33
2.6 Quality evaluation for scaling the variances (average variance shown on x-axis). 34
2.7 Quality Evaluation on the number of cluster label 35
2.8 Scalability runtime experiments for dimension size and db size. . . . 35
2.9 NORDS result on the wages data set. 37
2.10 NORDS result on the gene expression data. 38
2.11 NORDs result on the metabolic data. 39
2.12 Possible candidate subspaces for clustering. (a) Example 4D data set projected to two 2D axis-parallel subspaces. (b) Major eigenvector space (upper) and minor eigenvector space (lower) found by PCA. (c) Two arbitrarily oriented independent subspaces found by ISAAC. 43
2.13 Subspace S_{AB} is dependent to S_{CD}, thus all objects are clustered in the same way. The mutual information between them is close to 1, even the variance and cluster shape may be different. 48
2.14 The clusters between subspaces S_{AB} and S_{CD} are independent as they co-exist in different independent subspaces. Even if the cluster shape is similar, the objects are grouped differently. 49
2.15 Two independent subspaces S_{AB} and S_{CD} 54
2.16 Automatic greedy search procedure of ISAAC 55
2.17 Clustering procedure of ISAAC 57
2.18 F1 value of synthetic data set *sync* 59
2.19 Cluster Quality. the first row: synthetic data set “sync” with four two-dimensional independent subspaces; the second row: the four independent subspaces and clusterings found by ISAAC. 60
2.20 Runtime evaluation on different database size on sync 61
2.21 Nine raw samples from the Dancing Stick Figures. 62
2.22 Four objects of different shapes (ball and cylinder) and colors (green and red) from ALOI. 62
2.23 ISAAC on breast cancer Wisconsin (diagnostic) data set 64

2.24 The weights of attributes making up axes in the 3 dimensional subspace (The weights of attributes are represented by colors) . . . 64
2.25 ISAAC on metabolic data set - three independent subspaces 65
2.26 The weights of attributes making up axes in 3 independent subspaces (The weights of attributes are represented by colors) 65
2.27 The means of the clusters detected by ISAAC (top) and ORTH2 (bottom) in two subspaces (Dancing Stick Figures data). ISAAC identifies clear upper- and lower-body perspectives. 66
2.28 The means of the clusters detected in the ALOI data by ISAAC (three subspaces, top) and mSC (two subspaces, bottom). ISAAC successfully identifies subspaces which partition color and shape. In addition, it finds subspace in which objects of similar sizes are grouped together. (*best viewed in color*) 67

3.1 Attributes and Social Networks of Users 72
3.2 Clusters in the Multi-Model Network of Figure 3.1 73
3.3 Tensor Construction: Five modes and one attribute. 89
3.4 Synthetic Runtime Experiments 92
3.5 Effect of the initial number of clusters K. 94
3.6 Effect of cutoff threshold τ 96
3.7 Evaluation criterias on Facebook, Google+ and Twitter Ego-Networks. 98
3.8 Runtime for large unlabeled real data sets 99
3.9 Two Differently Structured Sub-graphs. 102
3.10 All Possible Connections of Three-node Primitives 105
3.12 Multiple Primitives. 109
3.13 Coding scheme evaluation on synthetic data. 115
3.14 Coding scheme evaluation on real world data. 116
3.15 Syn1 with Two Stars and One Clique. 117
3.16 Coding Cost of Clusters. 118
3.17 Syn2 with Three Stars. 119
3.18 Graph Partitioning of Zachary's karate club Data. 121
3.19 Precision of Synthetic Data Sets. 124

4.1 Example of our concept for mining temperature time series in the region of Germany. (a) shows the grid layout of MERRA for Germany, (b) measures the temperature of four specific, connected regions in this grid over time; (c) calculates the correlation coefficient between these temperature time series. The four selected grids are colorized in yellow or red. Red if the coefficient is an outlier, yellow else; (d) illustrates the matching of this found outlier (red) regarding its spatial region. 130
4.2 Example temperature data for 17 neighboring geo-locations. Every line corresponds to one location. The green line corresponds on the left hand to a closely correlated region compared to the others, on the right hand it is an extreme outlier with even negative correlation. 133
4.3 Correlation VS. DTW and Edit Distance. Correlating the red and green temperature curves would remark (a) as most similar curves. Using DTW or Edit Distance would resolve (b) as the most similar as the covered area (yellow) is smaller. 135
4.4 Creating a tensor from precipitation, temperature and wind speed data. 138
4.6 Synthetic outlier data varying the number of manually changed values for the attributes temperature, precipitation and wind speed. 143
4.7 Tropical Cyclone classification results. 144

5.1 Overview of approaches to data analytics using relational database systems. Our system supports the novel *layer 4*, where data mining is integrated directly into the database core, thus, leading to higher performance. To maintain expressiveness, high-order functions (*lambdas*) can directly be passed as parameters to the database operators. 150
5.2 Simplified query plans for k-Means clustering using the operator-centric and SQL-centric approach to data analytics in databases. . . 152
5.3 Query translation and execution with relational and analytical operators. A SQL query is translated to an abstract syntax tree (AST) consisting of both relational and analytical operators. The optimizer can inspect both types of operators. This approach provides highest integration and performance. 165

5.4 k-Means experiments. From left to right: varying the number of tuples N, dimensions d, and clusters k. Default parameters: 4,000,000 tuples, 10 dimensions, 5 clusters, 3 iterations. 171

5.5 PageRank and Naive Bayes experiments. From left to right: PageRank using the LDBC SNB dataset, dumping factor 0.85, and 45 iterations. Naive Bayes experiment varying the number of tuples N. Naive Bayes experiment varying the number of dimensions d. 171

List of Tables

2.1 Quality measures F1-value and Accuracy for real world data sets. The caption of the data set provides [data set(size, dimensionality, number clusterlabel)]. The best results are bold. 36
2.2 Table of Symbols and Acronyms 46
2.3 difference coding cost 53
2.4 F1 measure on real-world data (with dimensions (n;m)). $*$ failed because of non-trivial bugs in the OpenSubspace implementation [145]. 63

3.1 Evaluation Overlapping Clusters of Synthetic Data Sets 91
3.2 Sizes of all used real data sets. 97
3.3 Distinguishing Graph Structure on Real Data Sets 116
3.4 Example of differing people in MCL and CXprime 122
3.5 Compression Rates (Bits) 123
3.6 Precision of Real Data Sets (%) 125

5.1 Datasets for k-Means experiments. 172

CHAPTER 1

Introduction

Large graph datasets of social networks from millions of users and billions of relationships between them; complex, high dimensional relation data of huge dabase systems and petabytes of streamed environmental sensor data generated everyday in the world. Creating benefit of such “big data” is nowadays challenge and predictor for the growth and welfare of our societies [135, 92, 54]. But “big data” does not only come in sheer volume, it is about the complexity of the data measured in four specific attributes called the four “V’s” [43][27]:

volume The actual amount of data, that still need to be complex enough. Data, that simply needs lots of physical space does not necessarily belong to the big data phenomenon.

variety The different types of structured and unstructured data that organizations can collect, such as vector-type data, graph data, timeseries data and multimedia data.

velocity An indication of how quickly the data can be made available for analysis; the in-and out stream of data.

veracity An indication of data integrity or fuzzyness and the ability for an organization to trust the data and be able to confidently use it to make crucial decisions.

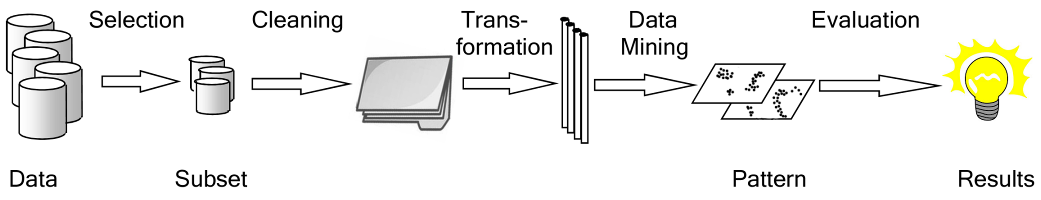


Figure 1.1: The knowldege discovery in databases process.

The science of extracting useful information from big data sets is usually referred to as "data mining", "machine learning" or "data science" (with slightly different focus [164]) and often the actual analysis of data is one step of the full Knowledge Discovery process. The Knowledge Discovery in Databases (KDD) process is a very interdisciplinary topic on the interface of the fields of statistics, machine learning and database systems. As the analyzed applications are of practical importance, there are now numerous books and surveys in the area [70] [181] [61] [63] [71] [67].

The KDD process is an iterative process depicted in Figure 1.1. It consists of data selection, data cleaning, data transformation and reduction, mining, interpretation and evaluation, and finally incorporation of the mined knowledge with the larger decision making process. The development of efficient computational approaches to data modeling (finding patterns), data cleaning, and data reduction of high-dimensional large databases. Methods from databases, statistics, algorithmic complexity, and optimization are used to build efficient scalable systems that are seamlessly integrated with the relational/OLAP database structure. This enables database users to easily access and successfully apply data mining technology in their applications.

As the KDD process is an essential model in data mining, we describe its individual steps in the following:

data selection The first step in the KDD process is about selecting which part of the data should be analyzed. In the easiest case this is done via queries inside a database. When the data is stored in separate files instead of a structured (relational) database, various problems emerge like redundancies and inconsistencies. Thus often a database for selecting and storing the data is chosen.

data cleaning Data cleaning is the process of consistently and fully combine data from different sources and conventions. This is the largest task in most KDD projects and as such takes a lot of time [71, 63]. Time, that can be greatly reduced by using a data warehouse, as the data is already cleaned in such systems.

data transformation This step performs a transformation of input data to generate output that can cope with the restrictions of the data mining algorithm that follows. Many algorithms are not able to process all kind of data types, which makes a *normalization or pruning* of the values necessary. This task is often combined with specific *attribute selection* in either manual or heuristic fashion for high quality results.

data mining The actual data mining describes the application of efficient algorithms (for database users: analytics), to find important patterns inside the provided data (sub-) set [67]. Typical data mining approaches are clustering, outlier detection, classification and association rule mining. We cover the most important ones for this thesis in detail in Section 1.1.

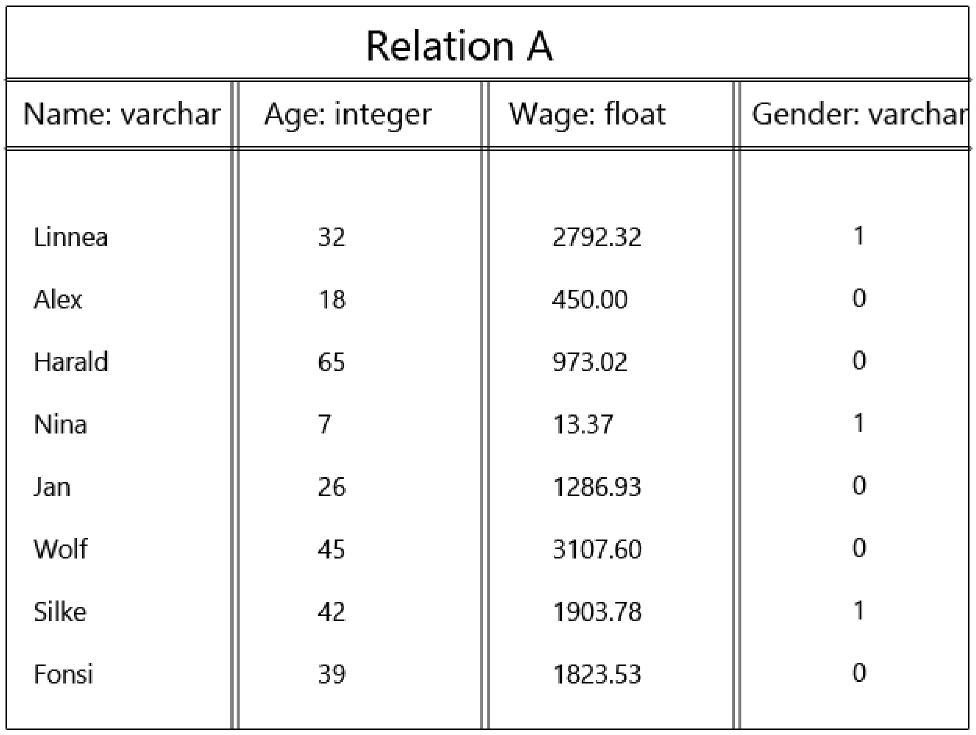
interpretation and evaluation The last step of the KDD-process is to present the results found by the system to domain experts, who will evaluate it further depending on the original goals. If these goals are not reached the process is iterated on any given step.

This thesis focuses mainly on the data ming step, by introducing new methods for data mining. This is done without leaving aside the importance of a database system, as an efficient system for processing data mining analytics.

1.1 Knowledge Discovery in Heterogeneous Data

As knowledge discovery is a wide and interdisciplinary field, such is the data and conventional use of names for such data. For clarity we explain the used conventions for naming different types of data throughout this thesis here.

1.1.1 Vector Data



Relation A

Name: varchar	Age: integer	Wage: float	Gender: varchar
Linnea	32	2792.32	1
Alex	18	450.00	0
Harald	65	973.02	0
Nina	7	13.37	1
Jan	26	1286.93	0
Wolf	45	3107.60	0
Silke	42	1903.78	1
Fonsi	39	1823.53	0

Figure 1.2: Vector Data: numerical (age, wage) and categorical (gender) attributes of a relation.

The term vector data stems from mathematics and the structure used in databases most frequently: relations. Thus, vector data is also called relational data. A relation or table is one possibility to get the unstructured structured, by imposing attributes or dimensions to a set of similar information (such like name, age or wage). In the relations of database system one tuple can be stored in vectors what may have caused the name of such data. Another term for a set of vectors or tuples would be a data matrix.

Vector data itself may include sub types: numeric values and categorical values. A numeric value is a value that is defined by a given number, and the number is the actual mathematical value like in the age or wage attributes of Figure 1.2. While categorical values indicate a given category like in gender where either "f" and "m" can stand for female and male or equally assigned "0" and "1", with "0" and "1" having a different meaning than their numeric counterparts. Algorithms that solve mixed data types in vector data, i.e. data consisting of numeric and categorical data types, are still relatively rare in state-of-the-art research despite their high occurence in database systems [162].

The concept of storing other collections of data (like lists) in a tuple as done in object-relational databases can not be modeled by classic vector data alone. Reasonably in this case already one tuple would need to be modeled as matrix, and the full data would need to be modeled as a tensor (n-dimensional matrix).

1.1.2 Spatio-Temporal Data

There are two extensions of plain vector data: spatial and temporal data. Spatial data relates to all data gathered from a geographic environment and as such have two special attributes (latitude and longitude) relating to the geographic position of this input data besides the remaining other attributes. Temporal data differs from standard vector data in the sense, that they have a timestamp (date) attribute and can be either found in a static or in a dynamic streaming setting. Several vectors with timestamps are called timeseries data. A vector set combining timestamps and geographic locations is called *spatio-temporal data*.

Mining such data has a much higher complexity than mining vector data alone [64]. The challenge consist largely of that these specific data or geo-location attributes need a special treatment compared to the other attributes of either numeric or categorical nature. In this thesis we focus on vector data first and later go on to a framework for large spatio-temporal weather data.

1.1.3 Graph Data

The structure of graph data lays emphasis on depicting the relationships among objects. This generates a possibility to model complex relationship types easily. Any kind of data can be modeled as a graph structure, with vertices representing data objects and edges expressing relationships between pairs of data objects. Moreover, in many application fields, like social networks, biology, commerce etc. modeling specific cases as a graph is the only sensible option.

We distinguish several different graph types:

undirected graph a set of nodes with two nodes being pairwise connected via one edge.

directed graph a set of nodes with connecting single edges. edges have a direction how they connect two nodes.

weighted graph either directed or undirected graphs with edges having a weight or distance.

attributed graph undirected or directed graph with nodes having relational attributes (combining graph data with vector data). Also known as property graph.

bipartite graph two sets of nodes all interconnected with the same type of edges.

multi-relational graph set of nodes connected with several types of edges.

heterogenenous graph several sets of nodes connected with several types of edges. Most complex and considered to be the model that depicts reality best. [87]

Due to the specific nature of data modelled as graph, mining a graph has an outstanding position compared to the ubiquitous vector or relational data. This is shown clearly in the analytics of graph data: While some graph mining has equivalent but somewhat differently names like community detection (equivalent to clustering in vector data) and anomaly detection (outlier detection), there are also tasks specific to graph data only like link prediction and node rankings. These task have in common that they are all linked to the unique structure of a network. For example finding the hubs (most outstanding vertice) or spokes of a graph [109], finding topological substructures like stars, trees, [175] link prediction for predicting the missing links out of the existing ones in dynamic networks[127], graph traversals [104]and many others[136][132].

In this thesis we show two new methods for community detection, one on undirected and the other for attributed graphs with overlapping communities.

1.2 Important Techniques and Systems

In this section we give an overview of all important techniques and systems mentioned throughout this thesis. We start with the techniques for data analysis.

1.2.1 Subspace Clustering

The general goal for clustering is to categorize data in several groups (cluster), in which data points (objects) in the same cluster are maximal similar and data points in different cluster are maximal dissimilar. Similarity measures are in most cases modeled as distance functions. Distance functions model similarity on the given attributes of an data object. The smaller the distance of two objects are, the more similar, the larger the distance the more dissimilar. Different algorithms use different distance function or other metrics to evaluate similarity. We consider model-based methods (like k-Means [131]) and density-based methods (like DBScan [62]) as the most widely known categories. The most important difference between model-based and density-based methodologies are that model-based algorithms have it hard to find arbitrary cluster shapes as depicted in Figure 1.3, but are usually more efficient to implement.

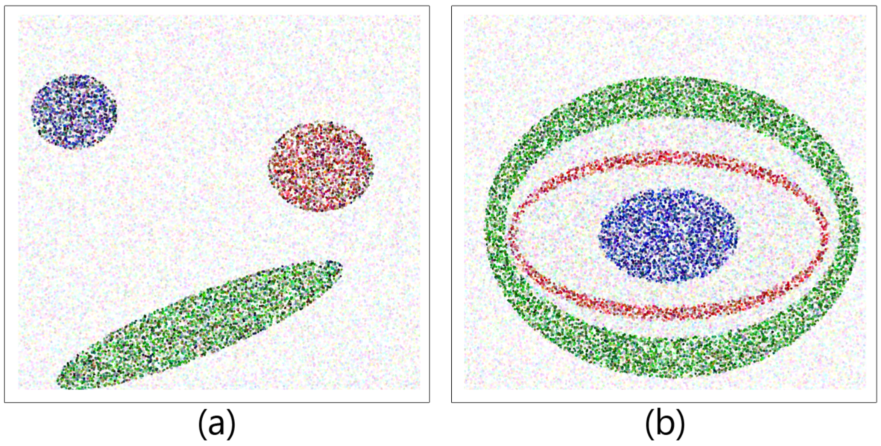


Figure 1.3: (a) axis parallel clustering shapes (b) arbitrary clustering shapes

In the paragraph before all clustering methods are working on full-space dimensions (all attributes are considered). But in this thesis we focus on *subspace clustering*. Subspace clustering tries to find clusters in several sub-sets of all dimensions, so called *subspaces*. Consider for example five dimensions A, B, C, D, E then a subspace clusterer would find two clusters in the dimensions A, B, C and three clusters in C, D, E. In this example the clusterings are overlapping in dimension C, non-overlapping (partitioning) subspace clustering is also possible.

Subspace clustering differs from full-space clustering that it has a major computational drawback: the so called *curse of dimensionality*. This drawback states, if the subspaces are not axis-parallel, an infinite number of combinations of subspaces is possible. Hence, subspace clustering algorithms utilize heuristics to remain computationally feasible, at the risk of producing inferior results [121]. For example, the downward-closure property (greedy bottom up-approach) can be used to build higher – dimensional subspaces only by combining lower-dimensional ones to result in a full space – an approach taken by most of the traditional subspace clustering algorithms such as CLIQUE [10] and SUBCLU.[107]. One drawback using such heuristics needs our special attention regarding the quality of the results of such heuristics in subspace clustering. Often times such greedy bottom – up or top-down approaches (dividing the full space into subspaces) generate a lot of redundant results, blurring the actually good results and in the worst case making such results impossible to interpret [24]. For this reason our methods focus on automatic redundancy reduction for axis-parallel and arbitrary subspace clustering.

1.2.2 Outlier Detection

"An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism" [122]. Outlier detection is closely related to clustering, as it depends on the cluster definition which objects do belong to a group and which do not and are thus regarded as outliers or noise. Still, it is not sensible to use the clustering approaches for outlier detection, as they are optimized to find clusters and not outliers. Accuracy of outlier detection depends on how good the clustering algorithm captures the structure of clusters. A set of many abnormal data objects that are similar to each other would be recognized as a cluster rather than as outliers or noise. Sample applications of outlier detection approaches include: fraud detection, credit card misuse, rare diseases etc.

In this thesis we give a new notion for outliers in an spatio-temporal environment of sensoric weather data regarding the given neighbors in the vicinity for one spatial data point in time.

1.2.3 Classification

Classification is a quite different approach from truly exploratory data analysis approaches like clustering and outlier detection. It works on the interface of data mining and machine learning, as parts of the data needs to be *labeled* (manually analyzed), to create a model that can be used for fitting the remaining data in an automatized fashion. Generally, the classification tasks can be divided into two parts:

1. training This part is about actually learning the given model on a training set of the full data. This training data is labeled so that for any given object the correct class label can be checked and predicted. The training part is usually solved by unprocessed training data.

2. testing In difference to the training part, we use the model created in training and mine knowledge out of the (remaining) data set. This data is called the test set and explicit knowledge is created with this data by predicting a label to a given object (it is not known whether this is true or false and solely depends on the model created beforehand). Basically we can only call this part "knowledge discovery".

The training and testing division is also used to gain classification accuracy. Clearly, this method is not applicable if the training data with known class label is very small. In this scenario usually the whole data set is used for training and test, and cross validation can be applied. Cross validation is a divide approach of the full data where always one part of the data is used for training in an incremental fashion. The evaluation of each increment follows afterwards.

Another typical accuracy measurement for classification is the confusion matrix, that shows which objects are classified into the correct class label. Logically this accuracy measure only occurs during the training phase as only here a gold label is available. For consistency reasons it is important to evaluate which input data is used for the test data. In the extreme example of taking the same data for test and training, the classification accuracy might be high, but this is erroneous. This effect is called overfitting.

Most well known approaches are Support Vector Machines [35], Bayes classifiers [126] which base on conditioned probabilities for attribute values of the different classes, and Nearest Neighbor classifiers which miss out of finding explicit knowledge but instead work directly on the training data. Decision Tree classifiers [16] deliver explicit knowledge by creating results in decision tree format. All featured approaches are usually evaluated on small training data sets that are stored main memory resistant.

In this thesis we show one new method for classifying spatio-temporal data and we show how to implement and integrate the algorithm Naive Bayes [126] in main memory database systems.

1.2.4 Minimum Description Length

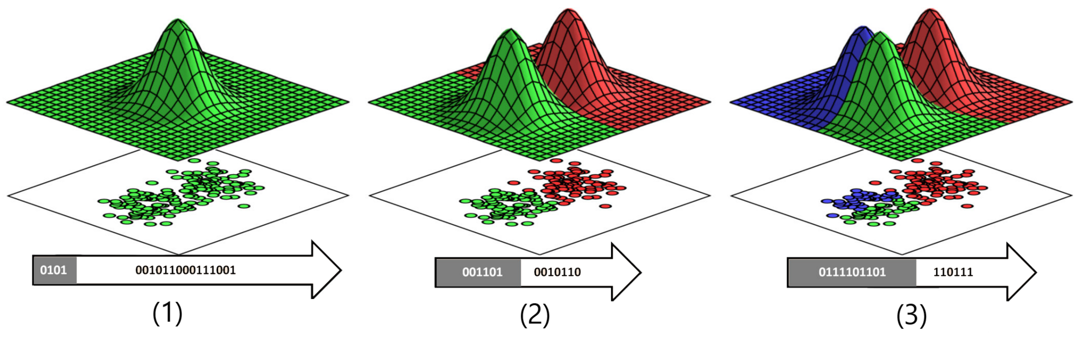


Figure 1.4: Minimum Description Length: Compressing the same data with three different models. The shortest bitstring in case (2) of model (grey) and data (white) yields the best representation. (Image Source [163])

Minimum Description Length (MDL) is a methodology deriving from information-theory that serves several important purposes for creating new methods in heterogeneous data. Its most general purpose is to generate a *measurable quality* by compressing the new model in combination with the given data set. Thus, such compression is called loss-less, as the full data set is included into the compressed bitstring. As example consider Figure 1.4. Here, three cases for compressing a model consisting of gaussian mixture models and the underlying data set are depicted. In case (1) a model of only one gaussian maps a given data set. As the model is easy to compress the more complex underlying data is not, thus requiring a much larger bitcode than the model. Case (2) shows the most balanced case where two gaussians are modelling the data set creating two ball-sized groups. The number of bits for model and data set is equal. Lastly, in case (3) the model is getting more complex using three gaussians, the model needs more bits to be represented than the data, resulting in a longer bit string than in case (2). Clearly the model best representing the data is case (2) as can be distinguished by the number of bits generated by MDL, where the shortest bitstring always represents the best model for the data. Thus MDL is called a model-selector that for any given method, be it clustering, outlier detection, classification or others shows which model best represents the given data. With that knowledge we are able to run algorithms without their hard to set input parameters, letting the method automatically choose the number of parameters needed.

In this thesis we show further ways of using the strength of MDL by creating own coding schemes for the given tasks. We use it as objective function (model selector) to determine which of our models best reduces redundancy for subspace clustering and community detection, automatizing our approaches by getting rid of all input parameters for the given use case.

1.2.5 Tensor Factorization

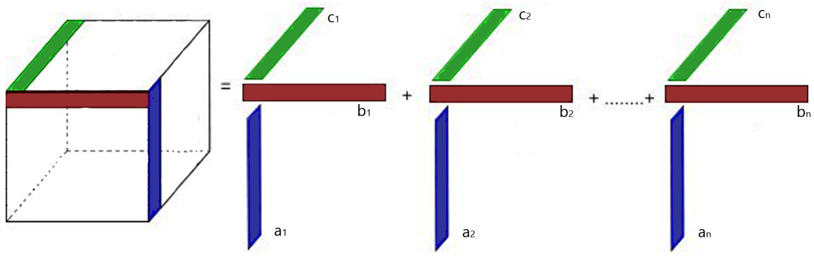


Figure 1.5: PARAFAC decomposition of a three-way tensor as a sum of n outer products (rank-one tensors).

Tensor factorization is in general a decomposition of a tensor into its latent factors (also known as concepts or components) [161]. Tensors are, essentially, multi-dimensional generalizations of matrices; for instance, a two dimensional tensor is a common matrix, and a three dimensional tensor is a cubic structure [161]. Each dimension of a tensor is called a ‘mode‘. For example in Figure 1.5 the cubic (3-mode) tensor is decomposed into n latent factors. The modes of the latent factors $(a|b|c)_1, .., (a|b|c)_n$ are each a weighted subset of the attributes in the original tensor.

To use a tensor for structuring the underlying data is especially useful for complex settings like in timeseries (spatio-temporal) data or in graph data. Tensors and tensor factorization are powerful tools to model and analyze data, and are increasingly gaining popularity in data mining and machine learning [119]. Tensor factorizations like PARAFAC [161] and Tucker3 [123] can be seen as a generalization of matrix factorizations, such as the Singular Value Decomposition [14] or Principal Component Analysis [42] for higher order matrices (tensors). As such, tensor factorization serves as a dimensionality reduction method of higher dimensional matrices. Recent research [157] has found a strong correlation between tensor factorization and co-clustering. The co-clustering problem seeks to simultaneously partition the rows and the columns of a matrix to produce "coherent" , overlapping groups called co-clusters.

In this thesis we apply the PARCUBE method [158] who enhances standard PARAFAC for very large data in an efficient manner. First, we apply it for data in a spatio-temporal context, hoping to cluster and reduce dimensionality the latent factors for several hypothesis. Second, we use tensor factorization as useful representation in an attributed graph clustering setting, by showing the network structure in the first two modes and the attributes in the third mode of the tensor and factorize them to achieve efficient co-clustering of such complex data structure.

1.2.6 HyPer

HyPer is our database system of choice, as it is a (former) research project with several outstanding attributes compared to other relational database systems that contribute to the efficiency of data mining algorithms. In the following we go into detail and show which general aspects of HyPer have the greatest impact on running data analysis algorithms in this system:

in-memory DBMS HyPer claims that most data fits into main-memory (nowadays up to several TB) [116], and with data already stored in main-memory the bottleneck of loading data from disk is easily avoided. Besides, even with already loaded data in main-memory, a disk-based systems is slower than a DBMS operating fully in main-memory. This helps complex data mining algorithms even further.

hybrid OLTP and OLAP DBMS HyPer belongs to the emerging class of database systems that have – in addition to an OLTP engine – capabilities to run OLAP queries directly on the transactional data – be it vector, geo-, or graph data [155][182]. This empowers data mining techniques in the sense, that the data is always updated and restructured and not staled. Other known hybrid systems include the database SAP HANA and its associated research prototype [66].

compilation framework A lot of overhead is avoided by HyPer on query execution by its compilation framework [148] enabling a fast runtime for data-intensive queries. In particular, by a special technique called *data centric code generation*, where SQL is put to assembly and then compiled instead of standard function calls done by the database's interpreter. Here, the query execution is divided into two phases: compiling and actual execution. For data exploration algorithms this means a strong increase in runtime as unnecessary branches are skipped by prior compilation.

multi-version concurrency control HyPer's multi-version concurrency control (MVCC) system [150] is exceptionally useful for long-running data exploration queries. It is similar to a timestamp-ordering approach, that keeps older versions of just changed tuples besides tracking which transaction changed which tuple. Especially complex workloads benefit from this kind of versioning, as their runtime is unaffected by other OLTP and OLAP queries.

instant loading Loading different data types into the analytics system takes a good amount of time compared to the actual analysis in several standalone systems. For readability and portability reasons data is often stored as plain text outside of database systems. For this purpose CSV files are commonly used. In traditional database systems, the intense parsing of these types of files constitutes a significant bottleneck in the loading process and thereby severely slow down the process of getting data into the system. HyPer managed to double the performance of this loading process and achieves "wire speed" with their instant loading system making the bottleneck of loading large data insignificant[143].

adaptive parallelism In several use cases including data mining analytics, some data is heavily skewed which creates a bottleneck for internal parallelization of one query. HyPer solves this by dividing the data in small *morsel* to apply adaptive threading [125].

In this thesis we show how data mining algorithms can be most efficiently and effectively integrated into a hybrid DBMS such as HyPer. We prove this runtime efficiency on three classical data mining approaches for graph and vector data: the clustering algorithm k-Means [131], the classification algorithm Naive Bayes [126] and the graph node ranking method Pagerank [34].

1.3 Contribution and Outline

All in all, the main contribution of this thesis is the creation of new state-of-art methods for analyzing a vast variety of data types, depending on the application these data derive from. For all novel methods creating new techniques like several specific coding schemes by Minimum Description Length (MDL) [80], elaborating new quality functions and specific modeling of the data was crucial besides combining existing approaches.

The remainder of the thesis is structured as follows:

Chapter 2: This chapter covers two methods that focus on creating non-redundant clustering results in complex high-dimensional vector data. The two methods differ in the sense what clusterings they are able to find: the first, NORD, finds axis-parallel clusters while the second is the natural extension of NORD, ISAAC, which does find all kind of clustering shapes, so called arbitrary clusterings. Both approaches are automized through minimum description length, which objective function does rely on non-redundant results for subspace clustering.

Chapter 3: This chapter covers two methods for graph analyzation, both on undirected graphs one without attributes and one with attributes on the nodes. As complexity rises strongly with the attributes, for larger graph data a dimensional reduction method with tensor factorization was applied to gain efficient throughput, where the first approach, CXPRIME, works with automized description length on minimal graphlets.

Chapter 4: While the first two chapter deal with clustering problems, this chapter shows a framework on large temporal-spatial data that solves a classification problem. Predicting natural hazards in global weather data is the goal in this study. We solve this by applying different existing or little varied techniques in a full framework and evaluate it. The framework combines data mining tasks with machine learning approaches.

Chapter 5: This chapter combines all other chapters in this thesis and shows how to create them and all other data mining algorithms efficiently as operators in the main memory database system HyPer. This is explicitly done by the example of other, traditional data mining approaches like K-Means [131] for clustering, Pagerank [34] for graph mining and NaiveBayes [126] for classification algorithms. Our approach takes HyPer specific architecture into account and enables our algorithms execution plan to be modified by the databases' optimizer.

CHAPTER 2

Methods for Vector Data

Parts of this chapter have previously been published in ICDM 2016 [203] and at PAKDD 2017 [99].

In this Chapter we focus on methods applied to vector data as explained in Section 1.1.1. We show two new methods on *numeric* data that are heavily optimized on reducing *redundant* behavior in subspace clustering. The first method, NORD, works on axis parallel data while the second method, ISAAC, expands NORD for arbitrary dimensions.

2.1 NORD - Non-redundant Subspace Clustering

Clustering in general is a powerful exploration tool capable of uncovering previously unknown patterns in data. Subspace clustering is an extension of traditional clustering, based on the observation that different clusters (groups of data points) may exist in different subspaces within a dataset. Subspace clustering has attracted a lot of attention because in many applications it is natural that objects are clustered flexibly in different subspaces. Typical applications for subspace clustering are e.g.:

- life science data, like genes under different experimental conditions [23] or in identification and classification of diseases [110]
- security and privacy in recommender systems [206],

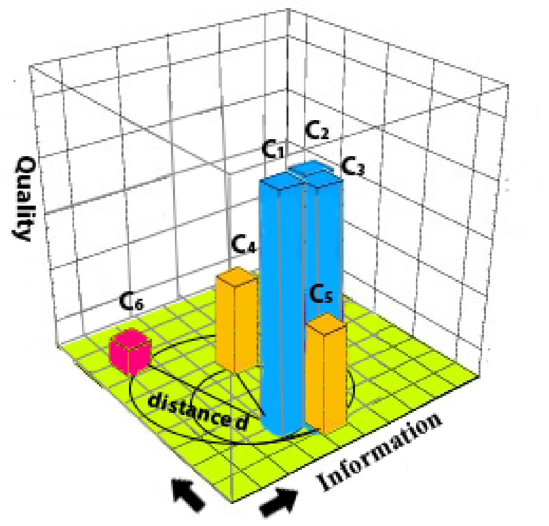


Figure 2.1: A possible redundant output of an unspecific subspace clustering method depicted with our new visualization technique. The clusters C_1 to C_5 show their redundancy due to their position on the information plane. The higher the distance d between the clusters, the less redundant they are to each other.

- computer vision, e.g. image/motion/video segmentation [108][47] as well as image representation and compression[198]

Allowing objects to be assigned to more than one cluster unfortunately not only yields interesting novel information which is valuable for interpretation but also can also contain a lot of redundant, already seen information. If the data contains a rich cluster structure in some subspace, all subspaces and all super-subspaces will also tend to exhibit some cluster structure. But is this information interesting? In most cases the answer is no. Provided that these clusterings are not superior in terms of quality, they are not interesting because they represent redundant information.

Consider the synthetic example in Figure 4.1. Here, we show a possible redundant output of some undefined (subspace-) clustering method with our visualization that is based on information-theoretic measures. The positioning of the six clusters (the bars) in Figure 4.1 is chosen such, that a pair-wise distance metric, which considers the set of dimensions and the set of objects of each clusters, is maintained. As the clusters C_1 to C_5 standing very close to each other, their novel information content is quite small. The higher the distance d the higher is the differing in the information content as for example in cluster C_6 Redundant results do only make sense when the quality of the clusters is superior to others.

What do we mean by *quality*? Our algorithm NORD relies on the information-theoretic idea of linking data mining to data compression. Any kind of non-random patterns can be exploited to compress data. The stronger the patterns and the better an algorithm succeeds in detecting them, the better is the compression rate. We define the quality of a cluster as its contribution to the overall compression rate.

The challenge is to distinguish interesting novel information from undesired redundancy. This is done by our novel clustering method NORD (for NOn-ReDundant) via information-theoretic measures that automatically balances the novelty and quality of a subspace cluster. The information we gain by a subspace cluster on our data is two-fold: we learn which attributes span the subspace of the cluster and we learn which objects belong to that cluster. Information-theoretic measures allow to quantify the amount of novel information provided by each cluster as well as its redundancy to the remaining clusters. We combine this idea with the quality measure to obtain a novel information-theoretic optimization goal for subspace clustering, that automatically balances both aspects.

The remainder of this subchapter is organized as follows. After giving an overview of our notation we define the characteristics of a clustering on a information-theoretic foundation to make it measurable in Section 2.1.1. Here, we also describe the visualization of our method that is closely connected to the defined quality of a cluster. Section 2.1.2 introduces the algorithmic concept. The evaluation of our algorithm NORD takes place in Section 2.1.3. An overview of the related work is given in Section 2.1.4.

Contributions

1. **Non-redundant Subspace Clustering by Balancing Quality and Information:** Besides optimizing the *cluster quality* we consider maximizing the *amount of non-redundant novel information in terms of cluster and subspace identification* as a second major optimization goal for subspace clustering, see Section 2.1.1. NORD is the first approach relying on information theory to make both aspects measurable in a comparable way such that they can be integrated into a common objective function.

2. **Efficient and Automatic Clustering:** Besides the parameters required for redundancy control the comparison methods require further parameters specific to the underlying clustering method, e.g., density thresholds. We also avoid such difficult to estimate parameters relying on the Minimum Description Length Principle. Section 2.1.1 discusses the overall optimization goal. We propose an efficient heuristic algorithm which automatically identifies the most interesting high-quality clusters, see Section 2.1.2.

2.1.1 Goal: Balancing Quality and Novelty

In this section we elaborate the theoretic foundation of our proposed algorithm called NORD. The first subsection, Section 2.1.1 defines the two correlated *quality measures* to assess the interestingness of a single subspace cluster: redundancy and information. In Section 2.1.1 we unify these measures into a single coding scheme. We formalize all measures from an information-theoretic perspective, which allows naturally balancing them in a single objective function that emerges in the overall optimization goal for our algorithm. These measures are paramount in our presented algorithm in order to assess the overall novelty of a cluster, i.e., the interestingness in terms of information given other clusters. Finally, in Section 2.1.1 we present a useful and elegant visualization technique for clustering results returned by NORD.

Notations

Definition 1 (Database). *Let $DB \subset R^d$ be a finite collection of d-dimensional vectors. We call $n := |DB|$ the database size, d the dimensionality of DB and we call the vector $\mathcal{S} := (1, ..., d)$ the set of all dimensions, or full space.*

Definition 2 (A Subspace Cluster). *A subspace cluster C is defined as a pair $C = (S \subseteq \mathcal{S}, O \subseteq DB)$, where $C.O \subseteq DB$ is a set of data points existing in a cluster C of subspace C.S.*

Definition 3 (Subspace Clustering). *A subspace clustering $\mathcal{C} = \{C_1, ..., C_k\}$ is a set of k subspace clusters such that $\forall C_i, C_j \in \mathcal{C}, C_i \neq C_j : C_i.S = C_j.S \Rightarrow C_i.O \cap C_j.O = \emptyset$.*

Making Cluster Information Measurable

In this section, we present the formal definitions exploited by NORD: novelty, redundancy and overall information of a subspace cluster.

The information provided by a subspace cluster on the data is two-fold: Information about which objects are contained in the cluster and information about which dimensions span the corresponding subspace, more specifically:

Definition 4 (Information of a Subspace Cluster)**.** *The amount of information a subspace cluster $C = (S, O)$ provides on the data is:*

$$H(C) = H(O) + H(S), \;\; \textit{where} \tag{2.1}$$

is the number of data points in a subspace cluster C and $|DB|$ is the number of vectors in the full data set then $H(O)$ denotes the entropy of the cluster labels, i.e.

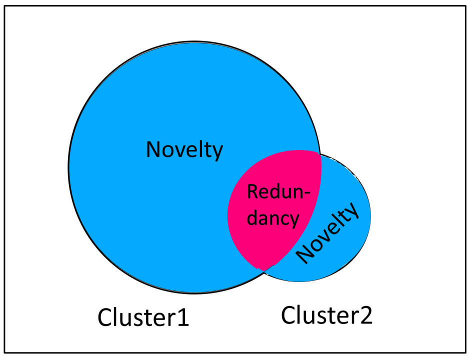
$$H(O) = -\frac{|O|}{|DB|} \cdot \log_2 \frac{|O|}{|DB|} + \frac{|DB \setminus O|}{|DB|} \cdot \log_2 \frac{|DB \setminus O|}{|DB|},$$

and analogously for the dimensions.

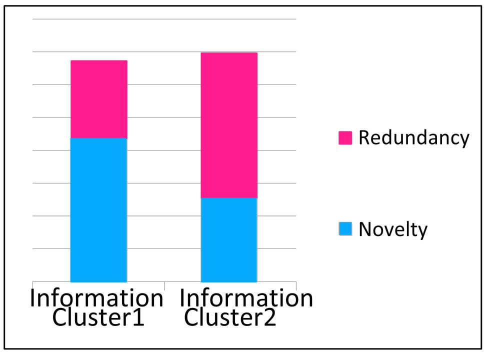
H(C) is a measure which quantifies in bits how much novel information cluster C provides on our data set. As a subspace cluster represents a subset of objects and dimensions of the data space, H(C) is the sum of the entropies of object- and dimension-assignments.

The information of multiple subspace clusters can be highly redundant. High redundancy can be due to the fact that multiple clusters are composed by similar subsets of the objects and/or reside in similar subspaces of the data space. It is therefore interesting to quantify how much non-redundant information a single cluster contributes to the overall clustering result.

The information of a subspace cluster thus usually consists of a novel part and a redundant part as can be seen in Figure 2.13. This figure represents the information of a single cluster by a corresponding circle. The information shared by both cluster is called redundancy, while the information shared by no other cluster is called novelty. It is clear that a single cluster may have a high information, but in the context of many similar clusters, it's novelty may be much smaller. We formally define the amount of non-redundant information among two subspace clusters as follows.



(a) two cluster



(b) Information of both cluster

Figure 2.2: The information content of two cluster. The overlapping part of cluster 1 and cluster 2 is redundant, the other parts are novel information provided by this one cluster. Together, novelty and redundancy form the full information of a cluster.

Definition 5 (Non-redundant Information between Two Subspace Clusters)**.** *The amount of non-redundant information of two subspace clusters $C_1 = (O_1, S_1)$ and $C_2 = (O_2, S_2)$ is defined as:*

$$NR(C_1, C_2) = VI(O_1, O_2) + VI(S_1, S_2), \text{ where} \tag{2.2}$$

VI denotes the Variation of Information, i.e. $VI(C_1, C_2) = H(C_1) + H(C_2) - 2I(C_1, C_2)$ with $I(C_1, C_2)$ is the mutual information of both clusters $I(C_1, C_2) = \frac{|O_1 \cap O_2|}{n} \cdot log_2 \frac{|O_1 \cap O_2|}{|DB|} \cdot \frac{|O_1|}{|DB|} \cdot \frac{|O_2|}{|DB|}$

NR is a metric, since it is the sum of two metrics. In [137] the authors used VI as similarity measure for comparing the results of different approaches to traditional partitioning clustering and proved that it is a metric. We use the basic concept of VI to quantify the similarity of the information which two subspace clusters from the result set of our method provide on the data. A NR of zero among C_1 and C_2 implies that $VI(O_1, O_2)$ and $VI(S_1, S_2)$ are zero, i.e. both clusters are composed of stochastically independent subsets of objects and dimensions of the data.

Encoding Quality

We follow an information-theoretic perspective to quantify the quality of a subspace cluster. To measure the quality of a subspace cluster we regard a subspace cluster as a pattern which we can exploit to compress the data. The more information (in bits) we can save by having identified some subspace cluster C in the data, the higher its quality. We first define how we can encode the set of objects O belonging to some subspace cluster $C = (O, S)$.

We model the data of a subspace cluster $C = (O, S)$ by a probability density function $pdf_C = No(\mu_C, \sigma_C)$. In a nutshell, the coding cost of a subspace cluster C corresponds to the deviation of the subspace clusters points from its expectation. Formally,

Definition 6 (Coding cost of a Subspace Cluster)**.** *Let $pdf_C = No(\mu_C, \sigma_C)$ be a probability density function. The coding costs of a subspace cluster $C = (O, S)$ consist of two parts, data costs dc and parameter costs pc, i.e. $cost_{pre}(C) = dc(C) + pc(C)$ with*

$$dc(C) = \sum_{o \in C} \log_2 \frac{1}{p(o, pdf_C(\pi_S(o)))} \text{ and} \tag{2.3}$$

$$pc(C) = \frac{|params|}{2} \cdot \log_2 |O|,$$

where $\pi_S(o)$ is the projection of object o to subspace S, params is the set of parameters of pdf_C, $p(o, pdf_C(o))$ is a function that returns the probability that the distance between o and a random point sampled from pdf_C is greater than the distance between o and the expectation $E(pdf_C)$ of pdf_C, formally:

$$p(o, pdf_C) = \int_{x \in S} pdf_C(x) \dot{I}(dist(o, x) > dist(E(pdf_C), x)),$$

$$E(pdf_C) = \int_{x \in S} pdf_C(x) \cdot x. \tag{2.4}$$

The data cost dc is provided by the negative log-likelihood of all associated points w.r.t. the PDF. The parameters of the PDF are determined after projecting the objects into the subspace S of the cluster. The better the cluster model fits the data, the smaller is this term. By using the dual logarithm we obtain the coding length in bit. In $pc(C)$ the first term represents the costs required to encode the $|params|$ model parameters of the pdf, in case of a spherical Gaussian pdf we have $|p| = 2 \cdot |O|$. Following central results from information theory [137][169], we use $\frac{1}{2} \cdot \log_2 |C|$ to represent a parameter. The second term represents the costs required to encode the cluster assignment and the last term the costs to describe the subspace of the cluster.

Example 1. *Let $C = (O, S)$ be a subspace cluster in a single dimension S, containing the set of points $O = 2, 2, 3, 9$. Let pdf_C be a uniform distribution in the interval $[0, 10]$. We obtain $E(pdf_C) = 4$. Following Equation 2.4 we obtain $p(2, pdf_C) = 0.6, p(3, pdf_C) = 0.8$ and $p(9, pdf_C) = 0.1$. Following Equation 2.3*

$$dc(C) = \log_2 \frac{1}{0.6} + \log_2 \frac{1}{0.6} + \log_2 \frac{1}{0.8} + \log_2 \frac{1}{0.1} =$$

$$0.737 + 0.737 + 0.322 + 3.322 = 5.118.$$

We see that the majority of the coding cost of C is contributed by the outlying point having value 9. Intuitively, we have to add more information to explain *this deviation from the expectation. We argue that the data cost dc can be used as indication of the quality of the conciseness of a subspace cluster. At the same time, we obtain*

$$pc = \frac{2}{2} \cdot \log_2 |O| = 1 \cdot 2 = 2,$$

thus yielding a total

$$cost_{pre}(C) = dc(C) + pc(C) = 7.118.$$

In the remainder of this chapter we use a spherical Gaussian PDF requiring $|p| = 2 \cdot |S|$ parameters (mean, variance) but our coding scheme can be extended to support different distributions and rotated clusters. Intuitively, the smaller $dc(C)$ the better is C. However, it is difficult to judge $dc(C)$ since without a reference coding cost, the absolute value does not say much. Therefore, we first introduce a baseline coding cost for *unclustered objects* and dimensions before defining the quality of a cluster as the saved costs over this baseline.

Definition 7 (Baseline Coding Cost)**.** *The baseline coding cost for some unclustered data set U is as follows:*

$$db(U) = \sum_{u \in U} log_2(\frac{1}{pdf_{DB}(u)}). \tag{2.5}$$

The baseline is unimodal with parameters estimated from the complete data set DB and therefore denoted by PDF_{DB}.

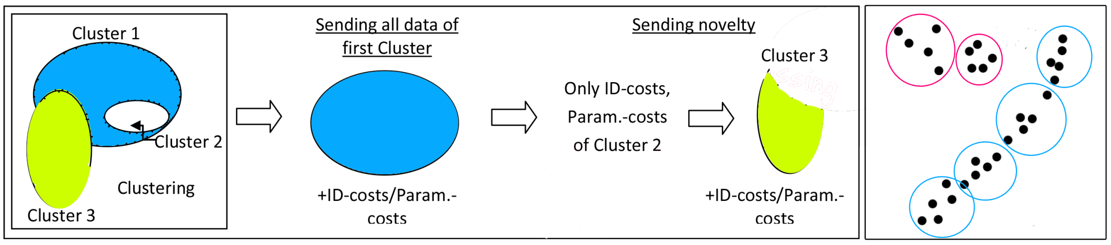


Figure 2.3: Figure Left: Sending order of the clusters of a clustering to a receiver. While the first cluster is sent in total, all subsequent ones are only sending their novel information. Figure Right: Estimating an arbitrary oriented cluster by microclusters with $m = 5$.

It is useful to compare the baseline coding cost of the full data set DB in full-dimensional space $\mathcal{A}$ which models all data as one unimodal PDF in comparison to the coding costs of any intermediate or final subspace clustering result. Subspace clustering only makes sense if we can improve on the baseline coding cost, otherwise we have evidence that our data does not contain any subspace clusters which can be represented by our cluster model. It is also useful to consider the baseline coding cost of a subspace cluster to define its quality in a comparable way:

Definition 8 (Quality of a Subspace Cluster)**.** *The quality of a subspace cluster $C = (O, S)$ is provided by the savings in bit over the baseline:*

$$Q(C) = db(U) - cost_{pre}(C). \tag{2.6}$$

To facilitate the comparison of the quality of different clusters, we often also consider the normalized quality $\hat{Q}(C) = \frac{Q(C)}{|O|}$ which represents the average number of bit saved per object over the baseline.

Balancing Quality and Information

Having formalized quality and information, we exploit the Minimum Description Length (MDL) [170] for coping with two challenges: We not only aim at automatically selecting the number and dimensionality for the subspace clusters but also aim at balancing among information and quality of the clusters. In other words, the second aspect means that the better the quality of a subspace cluster is, the more redundancy we allow w.r.t. the remaining clusters in terms of object assignment and subspace identification. The more novel information a cluster provides the lower its quality can be.

Clearly, information and quality need to be considered together: Only optimizing quality leads to the well-known problem of redundancy in subspace clustering, see Section 2.1.1. Only considering the amount of novel information, a completely non-sense result consisting of randomly selected subsets of the objects as clusters residing in subspaces of randomly selected dimensions would be optimal for any data set. The MDL principle avoids this by relating data mining to data compression. Imagine the data needs to be transmitted from a sender to a receiver via a communication channel. If the data contains patterns, in our case subspace clusters, we can exploit them to compress the data. More specifically, we minimize the total length of a two-part code: (1) The data cost representing the deviations of the data from the model, and (2) the parameter costs, representing the costs for encoding the model itself, i.e. the clusters, the cluster assignment and the corresponding subspaces. We formalize the description length of a subspace cluster before providing the overall objective function.

Definition 9 (Description Length of a Subspace Cluster)**.** *The full description length of a subspace cluster C extends the coding costs $cost_{pre}(C)$ in Def. 6 for extra parameter costs pc describing the cluster object assignments and the cluster subspace assignments.*

$$cost(C) = cost_{pre}(C) + |O| \cdot \log_2 \frac{n}{|O|} + |S| \cdot \log_2 \frac{d}{|S|}. \tag{2.7}$$

This coding scheme naturally balances quality and information: For a result consisting of many high quality but very redundant clusters, the data costs are low but the parameter costs are unnecessarily high because of many similar subspaces and clusters, and therefore the MDL score is far from its optimum. The result with random non-sense clusters has excessively high data costs due to low cluster quality and therefore scores very bad. However, this basic coding scheme encodes each cluster separately and does not consider any dependencies among clusters. However, since we already know that dependencies exist, since objects and dimensions can be assigned to multiple clusters, we should also exploit this information for more compact coding. Since the clusters need to be encoded in some sequential order to be transferred from a sender to a receiver over a communication channel, only the coding costs of the last cluster correspond to its novelty. In all other cases, we do not have the information of all other clusters available. How to efficiently find a useful coding order? We perform a greedy heuristic approach based on the VI-metric.

We rank the clusters according to their overlap in terms of mutual information of the object assignment and start the encoding with the cluster having the highest entropy. Since this is the first cluster, we need to encode it with its information, cf. Equation 2.1. In the next steps, we always select that cluster which has the minimal VI to the previous cluster as the next cluster to be encoded. We encode its ID-information with a bitstring corresponding to the novelty of the cluster w.r.t. all previously encoded clusters. If subspace clusters overlap in the dimensions and the objects, the overlapping part of the data is coded multiple times if we assume independence which is not necessary. Instead, it suffices for each overlapping object and dimension to encode the deviations from the previous cluster in our sorting order as depicted in Figure 2.3.

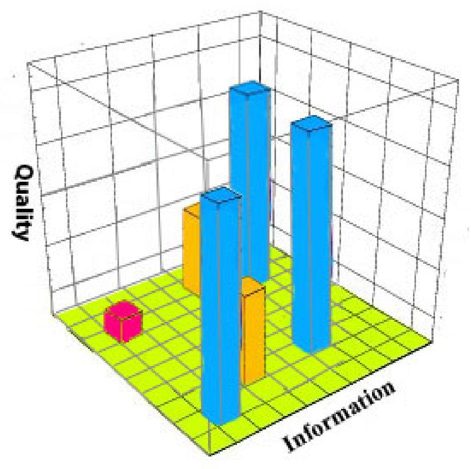
Last, we define the overall optimization goal for our subspace clustering which corresponds to minimizing the overall cost of the sorted cluster sequence:

Definition 10 (Overall Optimization Goal)**.** *Our optimization goal is to minimize the coding cost of the overall subspace clustering result, i.e.*

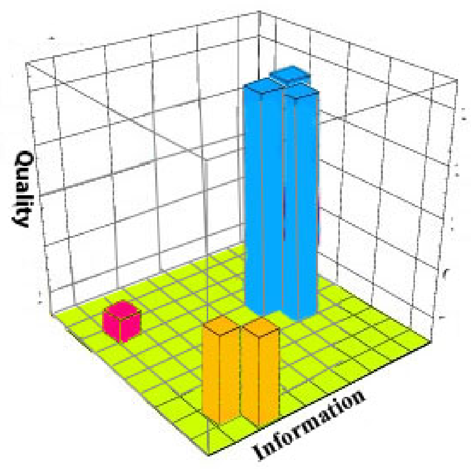
$$\min \sum_{C_i \in \mathcal{C}} cost(C_i | C_j <_s C_i).$$

Interpretable results

We explain our visualization on a synthetic example dataset with originally 10 dimensions with its result depicted in Figure 2.4 . This example refers to six clusters: Three high quality clusters in three dimensional space $\{D_0, D_1, D_4\}$, two clusters of lower quality in the dimension $\{D_2\}$ and the remaining unclustered dimensions with very low quality $\{D_3, D_5, D_6, D_7, D_8, D_9\}$. The quality of the clustering as defined in Definition 8 is scaled on the y-axis (the height of the bars), while the plane spanned by the x-axis and z-axis depicts the dimensionality-reduced Variation of Information (VI) matrix of the clustering. We apply Multidimensional Scaling (MDS) to embed the VI-Matrix. By doing so, the *position* of a cluster to the other clusters shows the level of similar information content in-between the different clusters. The spatially closer a cluster is to a respected other cluster. While the more similar data points and dimensions they share, the farer away they



(a) Novelty in Object Assignments



(b) Novelty in Subspaces

Figure 2.4: Visualization of quality and novelty on a synthetic example data set. Separated for the objects and the subspaces of the clustering. While the clusters seem correlated in the subspaces, their objects differ a lot. Best viewed in color.

are on the plane, the more different they are. To facilitate the illustration, we show our running example in two separate figures, one for object novelty and one for subspace novelty. In both figures, the cluster of lowest quality colored in red corresponds to the unclustered objects. The three high quality clusters seem quite related in Figure 2.4b as they are spatially close in their subspace, but as Figure 2.4a tells, their objects differ quite a bit. Usually NORD creates a visualization that combines object and subspace novelty in one plot, as can be seen in all other illustrations in this chapter.

2.1.2 Algorithm: NORD

After formalizing our quality functions and how to optimize the compression, we will now explain the algorithm of our approach in detail. Our algorithm is a greedy bottom-up approach divided into two phases: First, the initialization phase for creating the microclusters and second a recursive refinement step in which the initial quality of the clustering is improved by a) combining these microclusters b) removing redundancy by choosing and merging the most similar clusters first.

Initialization phase

Our initialization phase – although quite simple – contains one of the main concepts used in our algorithm and provides the algorithms' flexibility to find arbitrarily-oriented clusters: creating so called *microclusters* similar to the one in [186].

The idea is to create – in this case spherical gaussian - clusters as small as reasonable in terms of runtime and quality to later on merge them according to our quality function. Combining these microclusters even enables us to find arbitrary-shaped clusters as illustrated in Figure 2.3 on the right. The creation of these microclusters is described in detail in Algorithm 1. Here, the crucial part is to set and fit the inner parameter m – the number of minimal data points in one microcluster – according to the given data set DB. As a preprocessing step we initially set m to n, the overall number of data points, and recalculate the exact m iteratively depending on how many cluster are necessary to have at least the minimum of data points m included in a microcluster. Also, the result of our clusterer should have at least m data points per microcluster to ensure that our function works properly. The parameter m is significantly important for runtime and quality of the algorithm and its value is experimentally evaluated depending on the number of objects O in the experimental Section 2.2.4.

Subroutine 1. Initialization

```
Input: Numeric data set $DB$
1: for each one dimensional subspace $S \in S = \{s_1, s_2, \ldots, s_d\}$ do
2:     find the maximum number $k$ microcluster $M$, where $|M| > m$ in $S$
3: end for
4: return one dimensional microclusters $M$.
```

Raising Quality

With now every dimension of the data set consisting of very small microclusters the quality as defined in Definition 8 provided by our cost function is supposed to be quite low. Now, raising the quality in the first step considers only the quality *per dimension* but does not yet involve combining these dimensions. Hence, every single dimension is processed individually for raising the quality, by greedily merging the microclusters to their real underlying cluster structure as can be seen in Algorithm 2. In detail, the coding cost is calculated for every pair of clusters. Exactly, one entry in the cost matrix CM_{cc} corresponds to the MDL of two merged cluster minus the same two clusters separately compressed. The costs of two clusters separately include higher parameter and id-costs than the merged cluster. Since our clusters are usually initially way too small merging often pays off resulting in a negative entry of the cost matrix ($cc < 0$). If this is the case, the two microclusters will get merged to achieve a higher overall quality. The recursive merging stops when our quality function does not further improve.

Subroutine 2. Raising Quality Step

```
Input: Initialization result from Subroutine 1
1: for every dimension $d$ do
2:     Calculate the cost matrix $CM_{cc}(M, M)$ cf. Section 2.1.1
3: end for
4: while minimum coding cost $cc$ <0 do
5:     Merge the microclusters with minimum $cc$
6: end while
7: return one-dimensional high quality clusters $C$.
```

Merging Redundant Clusters

After merging every single dimension to its highest possible quality outcome, the dimensions are combined to higher dimensional subspaces. For this process we need a smart suggestion how to search through every subspace to not end up in exponential possibilities. This smart suggestion is provided by the information-theoretical concept of Variation of Information (VI). With VI we are enabled to measure novel information compared to other clustering results. The cluster pairs, providing the least novel information in terms of VI are the ones which are very similar to each other, thus not that interesting on their own. Those are the ones suggested by VI to be merged with each other. But a suggestion is not a proof that this would be a correct decision. The real decision whether it is a good choice to merge two - often multi-dimensional - clusters *globally* is done by our quality function. If the suggestions holds, which means in general the costs are decreased, the algorithm goes on recursively, if not, the merging is rolled back and other suggestions are tried until the algorithm finishes in a local cost optimum. The overall procedure is described in Algorithm 3.

Algorithm 3. Non-Redundant Algorithm NORD

```
Input: Numeric, high dimensional data set $DB$
Output: Overlapping label (optional)
 1: Initialization;
 2: for all one dimensional microcluster $M$ do
 3:    Generate the coding cost $CC$ as base cost $bc$ cf. Section 2.1.1
 4: end for
 5: Apply quality step cf. Subroutine 2;
 6: Re-generate $CC$
 7: while $CC$ decreases AND $CC > 0$ do
 8:    for every two (one dimensional) cluster do
 9:       Calculate symmetric VI-Matrix $VI(C_i, C_j)$ cf. Section 2.1.1
10:    end for
11:    Rank the minimal VI entries
12:    Select (next) minimum entry from $VI$
13:    Merge the clusters with highest redundancy
14:    Re-calculate $CC$ cf. Section 2.1.1
15: end while
16: Repeat several times until convergence.
17: Compute visualization cf. Section 2.1.1
18: return non-redundant Clustering $\mathcal{C}$.
```

Complexity Analysis

But the single steps of our heuristic approach are efficient: Our initialization step where the creation of microclusters takes place is determined by the runtime of our partitioning clusterer which is $O(nkdi)$, with k being the number of microclusters chosen and i the number of iterations. Then, for raising the quality of these one dimensional clusters needs to create the cost matrix M_{cc} for every combination of clusterlabel l, for which the MDL for every cluster combination is calculated. As the quality function from the MDL is linear in the number of objects O by exploiting the gaussian entropy, the overall runtime of this step is $O(l^2O)$. For the last step, the heuristic search via a matrix M_{VI} that holds the variation of information (VI) is also quadratic in the number of matrix entries. The VI itself needs a linear calculation over the number of clusterlabel l for the entropy H and a quadratic one for the mutual information I for the number of dimensions d. Therefore the overall procedure for creating a smart suggestion from the VI, costs $O(M_{VI}^2 l d^2)$.

2.1.3 Experimental Evaluation

In this section, we compare NORD with recent representatives of non-redundant subspace clustering paradigms like RESCU [144], INSCY [18]and STATPC [139]. For a fair comparison we implemented NORD in JAVA and used the evaluation frameworks OpenSubspace [145] and OutRules [146], WEKA extensions, where all mentioned competitor methods are implemented in JAVA as well. The frameworks also support standard parameter settings for each approach, which makes it easier to find the optimal selection of parameter for the comparison methods. All runtime experiments were done on the same machine, an Intel Core Quad with 3 GHz and 6 GB main memory.

Quality Evaluation Measures

Before starting with the actual evaluation, we chose to explain the used quality measures for our synthetic and real data sets as it has been quite a matter of debate what quality measure to use when. We measure quality as the *F1-score*, which is the harmonic mean of precision and recall and most of the time alone used for subspace clustering evaluation as can be seen in most of the related work [18][139]. The second used measure is *NMI* (normalized mutual information), an established quality measure for partitioning clustering and *accuracy* as used in classification. All of them have their advantages and disadvantages and are more or less adequate in different contexts. The F1-score (also called F-measure, F1-value etc.) measures the average purity of a clustering depending on the label which is most often appearing inside a cluster, while NMI measures the general agreement among given class labels and the cluster labels assigned by the algorithm. Since NMI has been designed for partitioning clustering, we cannot apply it in the real data experiments since usually only one labeling of the objects into classes is available and not multiple cluster labels as detected in subspace clustering. On the synthetic experiments we exactly know about the multiple cluster labels of each object and therefore can provide a comparison by slightly extending NMI: we compute the NMI considering each distinct *combination* of cluster labels as one class. We report the F1-score in the same way for the synthetic experiments. F1-score and NMI suffer from the fact that the subspace labelling (multiple label) is not comparable to projected clustering label used by these quality measures.

For that reason we cannot apply classification accuracy for showing a clustering quality result, but it gives an extra hint on the general underlying data especially in combination with the F1-score. The accuracy result is only vulnerable when the clusters are of very different sizes. Here it can be quite high for a very bad clustering (for example all data points are in one cluster, and the second cluster is very small). This is balanced by the F1-score, while accuracy on the other hand could balance the F1-score if the result would have a very high number of cluster. We believe with this selection of quality measures, a fair and thorough comparison of quality can be given.

Synthetic Datas

For the evaluation on synthetic data, all generated data sets correspond to variants mentioned in this subchapter. If the modifications are stronger it is explained in detail in the text. The only parameter of our method is the minimal number of

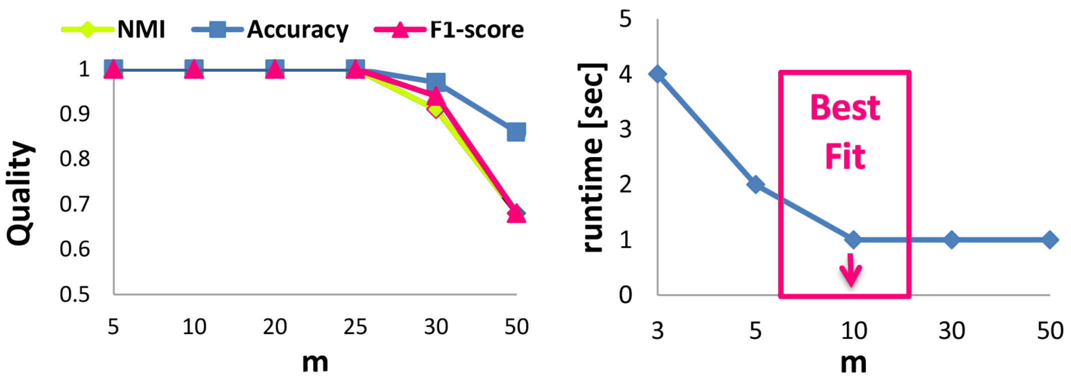


Figure 2.5: Quality and runtime evaluation of inner parameter m.

points inside a microcluster m which is used in the initialization phase of NORD inside the algorithm. It basically corresponds to the minimal cluster size that can be detected and should be set to a small value. However, depending on the dimensionality of the data set, it also should not be set too small in order to ensure that the parameters of the microclusters can be estimated.

In Figure 2.5 we show now experimentally that this parameter m is very robust in a wide range without affecting the quality and the runtime of the clustering algorithm. Here, Figure 2.5(a) shows that all quality measures stay at a stable level in the range of $m = 5, .., 35$. Note that 35 already represents 2% of the full data set of size 1500 points. Figure 2.5(b) shows that the runtime remains stable until using less than 10 points per microcluster. A smaller size is anyhow not desirable since we need to have some points to estimate the cluster model parameters. Therefore, we set m to a constant value of 10 for all following experiments, as a small value being stable in quality and runtime, and being far enough away from the critical mass (smaller than 5) for every data set size.

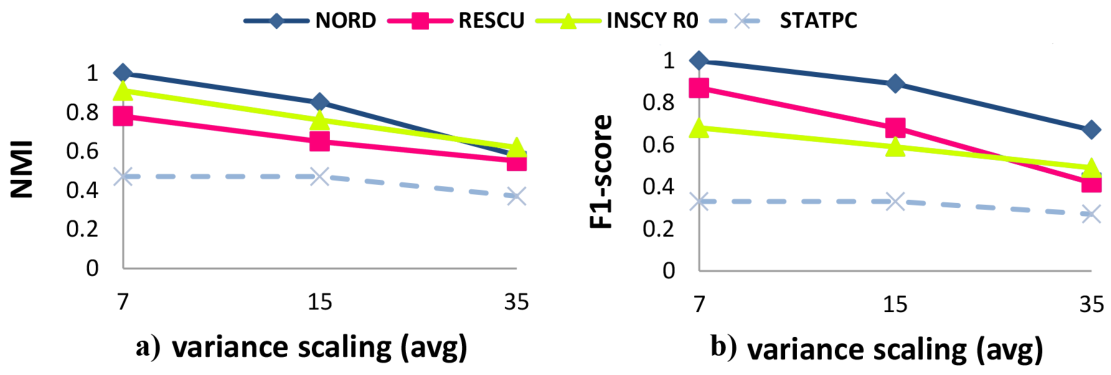


Figure 2.6: Quality evaluation for scaling the variances (average variance shown on x-axis).

Scaling the variance allows us to experimental evaluate the level of overlapping between different clusters and how the algorithm reacts to it. The variances are given in Figure 2.6 as the average variance of two clusters. For example if the plot shows a variance of 35, then this combines a higher variance of 50 and a lower variance 20. Figure 2.6 shows clearly that for all methods the quality decreases with a higher overlap. Depending on how redundancy is implemented and removed in the different approaches raises the quality. STATPC as approximate method has the biggest problems with the mutually overlapping data points that come with varying the variance of the cluster while NORD achieves the highest results in NMI as well as in F1-score.

We now study how the algorithms behave when increasing the number of cluster labels, which increases the number of subspaces containing clusters. Depicted in Figure 2.7 you see the scaling in range of [1,10] with 1 corresponding to each found subspace cluster having exactly one label in all subspaces like in projected clustering and 10 as an extreme case would result in having several clusters that each only occur in every single dimension for a 10 dimensional data set. We explain STATPC's extreme value in F1-score and NMI due to the fact that it creates a

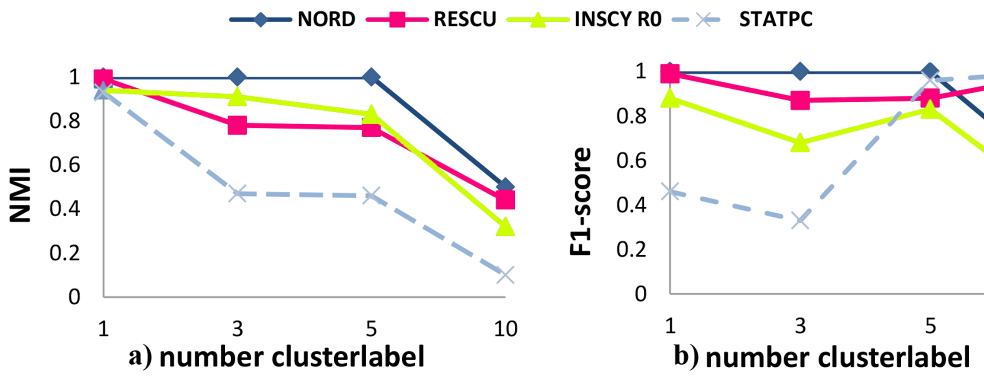


Figure 2.7: Quality Evaluation on the number of cluster label

clustering with many single clusters in full dimensional space. All others have it harder the more cluster label combinations are given, with NORD achieving the overall highest quality.

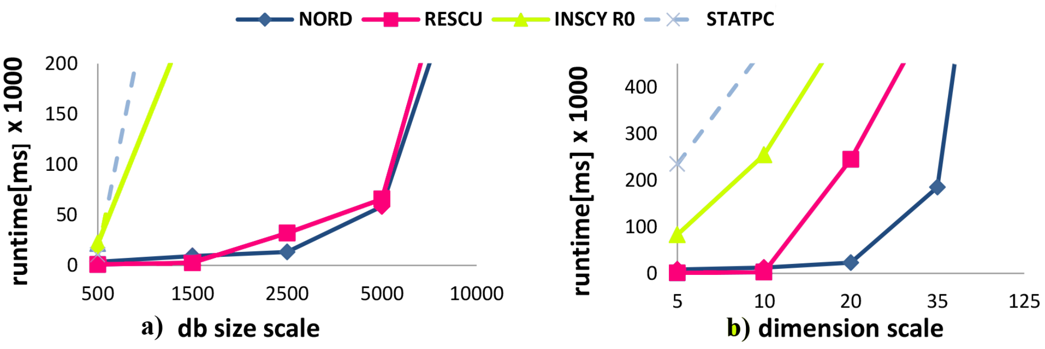


Figure 2.8: Scalability runtime experiments for dimension size and db size.

For scalability we show runtime experiments for db size (number of data points) and dimension size as well as quality evaluation for scaling the variances of our synthetic data sets. Figure 2.8 shows the plots for all comparison methods. While NORD scales similar with RESCU for the db size, it outperforms all other algorithms when scaling the dimensions of the data set. Besides the level of redundancy removal also seems to affect the runtime performance. STATPC as the only one having only approximative redundancy removal scales the worst compared to the fully non-redundant comparison methods.

Dataset	**Liver (345,6,2)**		**Shape(160,17,2)**	
Algorithm	F1-score	Accuracy	F1-score	Accuracy
NORD	**0.64**	**0.65**	0.56	0.56
INSCY	0.62	0.59	0.56	0.61
STATPC	0.57	0.58	0.31	0.62
RESCU	0.62	0.61	**0.60**	**0.73**
Dataset	**Diabetes (768,8,2)**		**Wages(534,10,2)**	
Algorithm	F1-score	Accuracy	F1-score	Accuracy
NORD	**0.71**	0.65	**0.68**	**0.72**
INSCY	0.58	0.65	0.27	0.66
STATPC	0.39	0.64	0.51	**0.72**
RESCU	**0.71**	**0.79**	0.66	0.65
Dataset	**Genes (4381,24,2)**		**Metab.(9584,43,4)**	
Algorithm	F1-score	Accuracy	F1-score	Accuracy
NORD	**0.66**	**0.86**	**0.93**	**0.86**
INSCY	0.33	0.45	0.77	0.76
STATPC	0.50	0.52	0.65	0.69
RESCU	–	–	–	–

Table 2.1: Quality measures F1-value and Accuracy for real world data sets. The caption of the data set provides [data set(size, dimensionality, number clusterlabel)]. The best results are bold.

Real World Data

For the real world data set evaluation we compare all comparison methods on six data sets as seen in Table 2.1 from different sources and various sizes. While *diabetes* is the original Pima Indian diabetes data from the UCI Machine Learning repository [19][1], the *liver* and *shape* data set are taken from the Opensubspace webpage [2]. They are originally drawn from UCI benchmark data sets as well. The data set *wages* is also publicly available on a webpage [3] and the *genes* data set belongs to the Spellman gene expression data available at the MINE projects webpage [4]. The metabolic data is a sensitive medical data and not available for free usage. Because all UCI data sets were also shown for INSCY and RESCU we applied the exact same parameters for these approaches that are given by their

[1]http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

[2]http://dme.rwth-aachen.de/en/OpenSubspace

[3]$http : //lib.stat.cmu.edu/datasets/CPS_85_Wages$

[4]http://www.exploredata.net/Downloads/Gene-Expression-Data-Set

authors[5]. The *diabetes* data was also one of STATPCs data and we could reproduce its result quite well by guessing the correct parameters. Despite of these data sets being already used by its competitors, NORD was able to outperform or at least get the same result (RESCU in liver) in both quality measures except for the shape data set where RESCU outperforms everyone else. For the other data sets we also guessed and used the default settings provided by OpenSubspace for the other algorithms, which was quite often a non trivial task due to the fact that STATPC has three parameters, INSCY seven parameters and RESCU applies even eight parameters to process redundancy accurately. With these methods being the state-of-the-art for redundancy removal in clustering, we can say that so far correct parametrization was crucial to process redundancy adequately. Now, the other data sets and their results are explained in more detail. The wages data that we

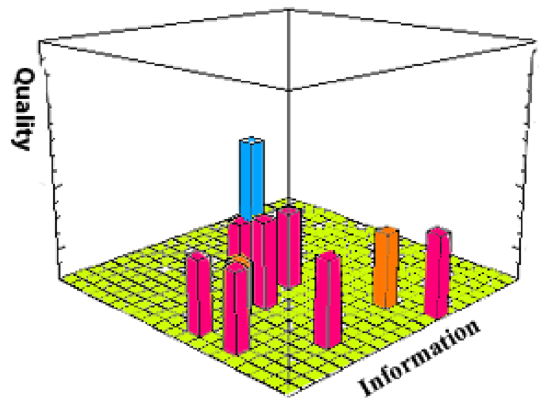


Figure 2.9: NORDS result on the wages data set.

derived from UCI Machine Learning Repository [19] consist of a random sample of 534 persons from the Current Population Survey (CPS). This social studies goal was to determine the impact of gender and other attributes like years of education, work experience and age on the wage. It provides information on wages and other characteristics of the workers, including sex, number of years of education, years of work experience, occupational status, region of residence and union membership. From all attributes only wage, age, work experience and year of education were numeric and thus relevant. The goal of the study was to determine (i) whether wages are related to these characteristics and (ii) whether there is a gender gap in wages. Our clustering algorithm NORD is able to find 9 meaningful clusters

[5]http://dme.rwth-aachen.de/de/OpenSubspace/RESCU

on this data set which are depicted in Figure 2.9. The blue cluster, with the highest quality shows the strong correlation between work experience and age. The three close cluster combine wage to age and experience. Year of education seems not interesting for the wage. RESCU had a very similar result with 10 clusters. STATPC scored well with 15 clusters also but INSCY did not manage to gain a high quality score with an F1-score of 0.27. The Spellman gene expression data

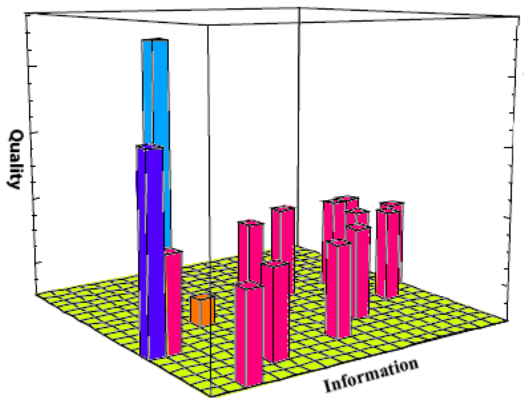


Figure 2.10: NORDS result on the gene expression data.

does study the mitotic cell cycle of yeast genes. It is also a quite well known data set for this community for the task to find functionally related genes using cluster analysis. In this relatively large data set (nearly 5000 data points), NORD finds 15 meaningful clusters depicted in Figure 2.10. Clearly, the two clusters with highest quality are also relatively similar to each other, both are containing 13 dimensions. RESCU could not process this dataset (and logically also not the metabolic data with nearly 10,000 samples). Even after we ran it on a machine with 48GB RAM, the JVM got an out of memory error. For INSCY we needed to modify this data set a bit, because it works with positive values only. The metabolic data set has been provided by one of our cooperation partners and contains the metabolic screening of 9584 newborns including children with metabolic disorders by genetic defects and a large healthy control group [22]. Besides the healthy newborns the data includes three metabolic diseases, the largest of them phenylketonuria (PKU) (305 newborns), MCC (44 newborns) and LCHAD with 60 individuals). NORD achieved a very good clustering result with an F1-score of 93% and 24 clusters depicted in Figure 2.11. The dark blue high quality bar here solely corresponds to the PKU disease (4 dimensions), clearly noticeable by the abundance of the

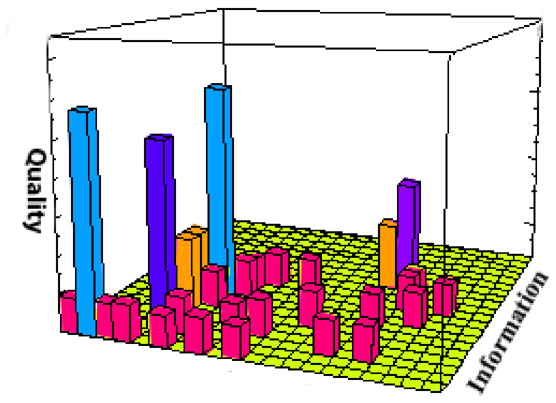


Figure 2.11: NORDs result on the metabolic data.

phenylalanine metabolite. The two clusters with the highest quality (light blue) both belong to the strong healthy control group(11 dimensions), in addition with most of the very small low quality clusters (pink, some orange). On one hand this makes sense due to the wide range of metabolites because of the strong correlation of different environmental and nutritional factors in the large control group, but is also a minor effect of existing noise and outliers. NORD also achieves to retrieve the small group of MCC patients (violet bar), and shows with that its ability to find clusters of very different sizes.

2.1.4 Related Work

This section provides the related work of the two fields that are connected by NORD - information theoretic clustering techniques and non-redundant subspace clustering.

Information-theoretic Clustering

Over the past decades clustering techniques have been studied extensively and been widely useful in statistics [96], pattern recognition [21] and machine learning [138]. Most of this research done in database community and others are clustering techniques in full dimensional feature space with parametrized algorithms like the famous K-MEANS [95] or DBSCAN [62]. Information-theory is often a method of choice to automize a clustering and make a clustering method parameter-free. We distinguish two types of the most common information-theoretic methods: One where we assign cluster labels to data points such that the mutual information between data and labels is maximized as in [38][65] and [57]. The other method is the in this algorithm also applied Minimum Description Length (MDL) that recently was regarded to have a significant impact on real datasets [165]. Famous algorithms applying MDL are for example RIC [29] which first applied different MDL criteria for arbitrarily oriented cluster despite to consider only axis-parallel ones in full dimensional space. Please acknowledge that due to limited space we can not regard all methods applying MDL that even seemed useful for graphs [129] and mixed data types [162]. For subspace clustering CLIQUE is the most famous one that first utilized information-theory for principal pruning to decide which cluster holds enough quality [10]. Newer techniques solve clustering for the subspaces of the data set in the hope for gaining more information as for example CURLER [186], which also inspired us to use simple objects (microcluster) to estimate the larger, arbitrary structure. In the following we focus on non-redundant *subspace* clustering techniques only.

Redundancy in Subspace Clustering

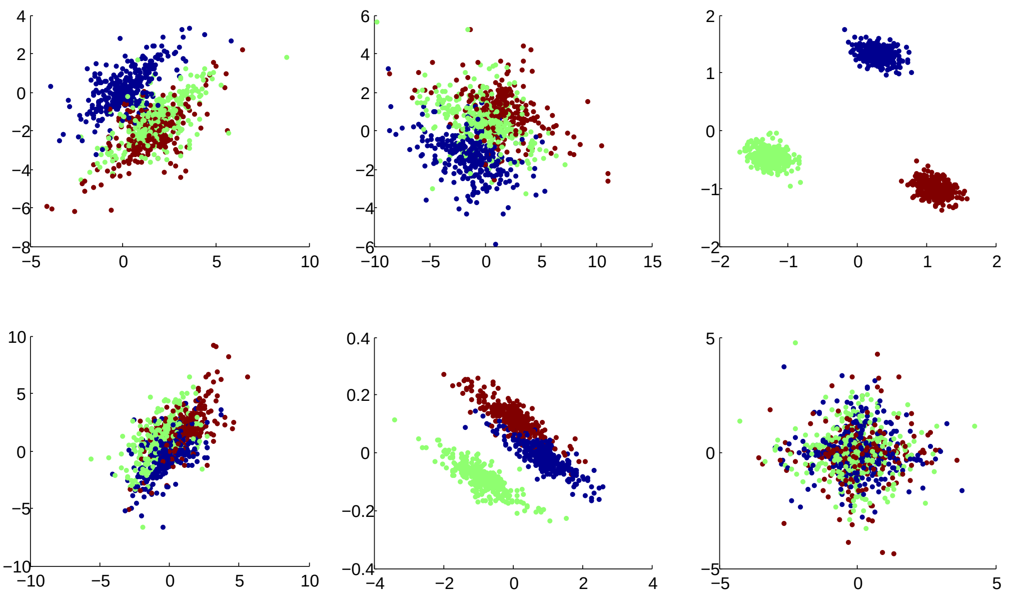
Redundancy is an efficiency problem in subspace clustering which was inherited from the very first bottom-up subspace clustering approaches like for example CLIQUE [10].In current research of subspace clustering it is a major optimization criteria to get rid of unnecessary redundancy. Recent research that focus on redundancy removal is the work of Mueller et. al. with RESCU [144] providing a new NP-hard model as optimization goal for non-redundant subspace clustering. RESCU also applies some simple cost function to measure the interestingness of a cluster that includes the density and number of dimensions of the cluster, but which is not balanced to the other traits of a cluster as our correlated quality notion. In RESCU as well as in INSCY [18] by Assent et. al. redundancy is regarded as lower dimensional projections that do not differ much from their higher dimensional counterparts. This relatively restricted redundancy definition absolutely gives reason to use a top-down splitting algorithm and an underlying tree-based index structure for increasing efficiency as done in INSCY. Another approach, NORSC [46] relates redundancy - as also done in RESCU - to coverage. NORSC defines redundancy depending on the count of (existing) data objects, thus the smaller the cluster the more redundant it might be. The algorithm STATPC [139] has a redundancy definition which we regard the most close to reality one (and thus used by us as well), because it defines not only the largest and highest dimensional cluster automatically as the best but states that there can be redundant projections in lower *and* higher dimensional space of the true (non-redundant) clusters found by statistical testings. Besides these most related approaches also quite some work is done on *orthogonal* non-redundant clustering: OSCLU [85] by Guennemann et.al. considers orthogonal non-redundant concepts. Improving OSCLUs efficiency by reducing its number of parameter is done by TSCLU [209]. Solving orthogonal non-redundant multi-view clustering with several alternative clustering solutions does Ying Cui et. al. [50]. Closely related to this paper is the work of Donglin Niu et. al. [153], whose method uses spectral clustering to gain multiple non-redundant clustering views. Besides this method using spectral clustering, all of the so far mentioned non-redundant approaches are density-based algorithms, that need a high number of input parameter. Specifically they need a density threshold that highly influence the clustering outcome. The work of Gondek et. al.[75] is focused on solving redundancy by using statistical models to evaluate it. None of these approaches is able to entire balance the redundancy with the novelty and information gain of the cluster. Besides so far only density-based or statistical methods do consider to handle redundancy in subspace clustering. Non-redundant Subspace Clustering for *categorical* data does exist in the work of He et al. [90]. The last non-redundant clustering technique we would like to introduce is the spare subspace clustering notion [59]. As a sparse representation (SR) used in this work is the smallest union describing the cluster, redundancy is naturally pruned but not weighted against the information content in the cluster nor arbitrary.

2.2 ISAAC - ISA Arbitrary Subspace Clustering

In real-world data clusters are not restricted to exist in axis-parallel subspaces but can be contained in arbitrarily oriented subspaces of various dimensionality. Most state-of-the-art approaches suffer one or more of the following drawbacks: They cannot explore all relevant subspaces efficiently and the interpretation of the result tends to be difficult since most existing approaches do not provide any visualization. In this subchapter we propose a new notion of subspace clustering: *independent subspace clustering*. We introduce the algorithm ISAAC (for ISA Arbitrary Clustering) which is the first method that combines independent subspace analysis (ISA) with generalized clustering. Generalized subspace clustering explicitly searches for relationships among the dimensions of a subspace cluster which are very interesting for interpretation. However, these relationships also exist in all redundant sub- and super-spaces bulking up the search space. ISA is very suitable to explore this large search space in an efficient way by partitioning the data space into subsets of subspaces exhibiting data distributions which are mutually statistically independent. However, ISA is a highly parameterized method. Finding suitable parameter settings is crucial for both clustering and ISA. Therefore, we establish an objective function based on the Minimum Description Length Principle automating both tasks. We prove ISAACs effectiveness and efficiency in our comprehensive experimental evaluation on synthetic and real data sets.

The main purpose of data mining and knowledge discovery is to find concepts, patterns, relationships, regularities, and structures of interest in a given data set [76]. Increasingly large data resources in life sciences, mobile information and communication, e-commerce, and other application domains require automatic techniques for discovering and gaining knowledge. One of the major data mining and knowledge discovery tasks is clustering, which aims at summarizing database objects such that similar objects are grouped together while dissimilar ones are separated.

In clustering real-world data, researchers face a variety of challenges called the 'the curse of dimensionality' which in most cases means distance measures (e.g. Euclidean distance, Mahalanobis distance, Manhattan distance) lose their effectiveness to measure similarity or dissimilarity when the dimensionality of the subspaces becomes infinite. Clusterers in those spaces cannot separate data into clusters well because the separation information of clusters is flooded. One kind of method to tackle this “curse of dimensionality” problem is to find several subspaces which contain most of the cluster-separating information but have a low dimensionality. Principle Component Analysis (PCA) [1] uses an orthogonal transformation approach to convert a data set of possibly correlated attributes into another data set of linearly uncorrelated attributes which are called principal components. The first principal component has the largest possible variance and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal (i.e., uncorrelated) to the preceding components [33]. By selecting several components to constitute a subspace, we can not only reserve most information but achieve dimensionality reduction. Independent Subspace Analysis [103] aims at linearly transforming original space into several independent subspaces. In the transformed space, subspaces are independent to each other and also reveal different structures hidden in the data set. Clusters in different independent subspaces are also independent to each other. In addition, these subspaces are not axis-parallel anymore because of the linear transformation. We use the term '*arbitrarily oriented*' to describe them.



(a) 2D projection (b) PCA (c) ISAAC

Figure 2.12: Possible candidate subspaces for clustering. (a) Example 4D data set projected to two 2D axis-parallel subspaces. (b) Major eigenvector space (upper) and minor eigenvector space (lower) found by PCA. (c) Two arbitrarily oriented independent subspaces found by ISAAC.

An example for finding possible candidate subspaces for clustering is given in Figure 2.12. Here, a 4D data set contains three clusters overlapping each other in axis-parallel space as depicted in Figure 2.12(a). Conventional subspace clustering methods cannot perform well in this data set, as the cluster information is contained in every dimension and the three clusters are hard to separate from each other. Hence, these axis-parallel, original subspaces are having a bad quality for clustering as clusters are most likely not found by any clustering algorithm. Figure 2.12(b) shows two major eigenvectors in the upper part and two minor eigenvectors in the lower part found by PCA. Clearly, in the major eigenvector space, PCA does not find enough relevant information to separate the three clusters completely. In the minor eigenvector space, PCA maximizes the variance of the data set to separate the green cluster from the other two clusters. Again, the quality of both found subspaces for clustering remains quite low. The reason PCA fails in such settings is because it is a *global* dimensionality reduction technique that finds *one* optimal representation for the *complete* set of points.[120]

The algorithm we introduce in this subchapter is named ISAAC. In Figure 2.12(c) we see that it is able to find high quality subspaces and to correctly identify multiple clusterings, that is, a different clustering in each of two different arbitrarily-oriented subspaces. We can see that the number of clusters in each clustering may be different, and also note that the number of dimensions per subspace may be different (in this example both subspaces have two dimensions). Importantly, ISAAC focuses on *minimizing redundancy between clusterings by maximizing independence between subspaces.* Practically this means that each clustering found by ISAAC in each independent subspace is highly informative and non-redundant. The lower subspace, for example, shows tight grouping of objects with *common* color and shape. In contrast, the upper subspace shows tight grouping of objects with *heterogeneous* color and shape. The upper subspace hence encapsulates different grouping behaviors, which is non-redundant information potentially leading to another valuable domain insight.

To arrive at such a solution, our method ISAAC combines Independent Subspace Analysis (ISA) [102] and clustering in one automatic framework. Through ISA we linearly transform the original space into several pairwise-independent (non-redundant) subspaces. We find the appropriate subspace cardinalities (required by ISA) using a greedy heuristic search that exploits the Minimum Description Length (MDL) principle. For practically finding the correlation clusters in each subspace, we use EM clustering with a hard-assignment of objects to clusters after each expectation-maximization step (MDL needs definite assignment of objects to clusters), although we note that our technique is agnostic to the exact algorithm used in this step. Again MDL is adopted to automatically choose the number of clusters in each independent subspace, making ISAAC parameter-free in theory and practice.

Contributions

- **Finding promising independent subspaces efficiently and effectively** Avoiding of tough choosing the estimators for independence, we use MDL to reveal the relationship between coding and independence. Subsequently, we develop a greedy search algorithm to parameterize ISA more effectively and efficiently to acquire more promising independent subspaces.
- **Full detection of informative, independent clusters** We solve the task of arbitrary subspace clustering in a heuristic way by applying ISA together with the EM algorithm integrated with MDL on the found independent subspaces. Clusters found in independent subspaces are highly informative and independent.
- **Automation supported by data compression** As parameters for the algorithm are tough to set unsupervised, we apply an advanced coding scheme which works for different ISA implementations and clusterers based on the MDL to make the algorithm work fully automatically.
- **Parallel clustering and interpretable results** ISAAC can parallelly cluster in each independent subspace to flexibly handle high dimensional data sets more efficiently. The results of ISAAC are interpretable, which is valuable for decision makers who are not familiar with data mining techniques.

This subchapter about ISAAC is organized as follows: Section 2 introduces some preliminary knowledge on generalized subspace clustering and independent subspace analysis. Section 3 demonstrates the core part of our approach: the theory behind generalized independent subspace clustering and the coding scheme for automating the algorithm. Section 4 elaborates the detailed algorithmic procedure of ISAAC. Section 5 evaluates ISAAC and related comparison methods on synthetic and real data set. In section 6, we briefly survey related work. The following Table 2.2 gives a list of symbols used in this subchapter.

2.2.1 Preliminaries

Generalized Subspace Clustering

Clusters in real-world data are not only restricted to exist in axis-parallel subspaces but can be contained in arbitrarily oriented subspaces of various dimensionality.

Table 2.2: Table of Symbols and Acronyms

Symbol	Definition
ISA	Independent Subspace Analysis
KDE	Kernel Density Estimation
W	the demixing matrix of ISA
d	the dimension of the data set
ns	the number of subspaces
ds	the dimension of a subspace
$\widehat{f_h}(x)$	estimated density by KDE
n	the number of objects
h	the bandwidth of KDE
IQR	the interquartile range
$\lvert c_i \rvert$	the number of objects in the i^{th} cluster
K	the number of clusters in the data set

Generalized subspace clustering aims at detecting clusters in arbitrarily oriented subspaces. Objects exhibiting a common linear or non-linear dependence among their attributes are assigned to a common generalized subspace cluster. More formally, the problem specification of generalized subspace clustering can be stated as follows:

Definition 11. *(Generalized Subspace Clustering) A generalized subspace clustering is a partition of a data set DS into K clusters $\{C_1, ..., C_k\}$. Each cluster is defined as a triple $C_i = (o_i, d_i, f)$, where $o_i \subseteq DS$ and $d_i \subseteq d$. f is an arbitrary statistical dependence value among attributes in d_i.*

The two challenges we address in this paper, i.e. parametrization and the curse of dimensionality, are not independent but interrelated. Approaches to generalized subspace clustering can be very successful on high-dimensional data where conventional clustering algorithms tend to fail due to the curse of dimensionality, but only if they are suitably parameterized. Finding suitable parameters for clustering high-dimensional data is even more difficult than for conventional clustering since additional parameters need to be specified. Typical required input parameters include for example the dimensionality of each subspace, a threshold for object density in subspace clusters, or a threshold on the significance of the statistical independence relationship among attributes. On low dimensional data sets, parameter selection can often be supported by inspections of the data set. For example by using scatter plots, or suitable parameters, which can be estimated in a trial and error fashion. Both options are often not feasible for clustering high-dimensional data due to the large number of dimensions, non intuitive parameters and the runtime of the algorithms.

Independent Subspace Analysis

Another important method related to our approach is the Independent Subspace Analysis (ISA) that was originally developed by Hyvarinen et al. [103]. ISA, an extension of the famous signal decomposition method Independent Component Analysis (ICA), aims at linearly transforming original space to several independent subspaces in such a way that maximizes the pairwise independence or minimizes the pairwise mutual information between two different subspaces. Differently speaking,

Definition 12. *(Independent Subspace Analysis) ISA transforms a given feature space into several subspaces, achieving that objects in different subspaces are* statistically independent *whereas objects in the same subspace are dependent.*

In detail, considering input data set X has d dimensions, then the goal of ISA is to find an invertible $d \times d$ demixing matrix W such that $WX = S = (S_1; \dots ; S_d)$,$S_i$ and S_j $(i \neq j)$ are mutually independent. The estimation of the demixing matrix W in ISA is equivalent to the minimization of the mutual information I between the estimated independent subspaces (S_i),

$$MI(W) = \min_{W \in GL(d)} I(S_1; \dots ; S_d) \tag{2.8}$$

where $GL(d)$ denotes the set of $d \times d$ sized invertible matrices. ISA needs to transform the data into white space, so that the data has unit variance. This can be achieved by using the eigenvalue decomposition of the covariance matrix, i.e. $V \cdot \Lambda \cdot V^T = \Sigma$, where V is an orthonormal matrix containing the eigenvectors and Λ is a diagonal matrix containing the eigenvalues of Σ. After that ISA determines the demixing matrix W by iteratively updating the rows of W with for example the update rule proposed in [103]. However, the original FastISA can only separate the original data set into equal-sized subspaces, which restricts its application. Another tradeoff of FastISA is that it converges to a local minimum and needs apriori knowledge of parameter settings like the number of subspaces. This was later enhanced by other authors like Gruber et al. [79] and Zoltan et al. [180].

2.2.2 Generalized Independent Subspace Clustering

Now, the main ideas derived from generalized clustering and Independent Subspace Analysis in Section 2.2.1 are combined to form an efficient clustering algorithm. As said before, the main goal of this approach is to find highly independent, arbitrarily oriented clusters in a fast, heuristic manner without using any input parameters that are hard to parameterize in advance. But how can statistical independence between subspaces help to reach the goal? To answer this question, we first elaborate how statistical independence does affect clustering. Later we will show how this approach can be automated.

Statistical Independence in Clustering

Statistical independence is a broad concept and implicitly used in many data mining applications [191]. As ISA partitions the full feature space into several distinct independent subspaces, an independent subspace must have specific attributes that are crucial for clustering.

Definition 13. *(Independent Subspace) Two arbitrary subspaces S_i and S_j $(i \neq j)$ are called independent from each other when their mutual information is 0, i.e. $I(S_i, S_j) = 0$.*

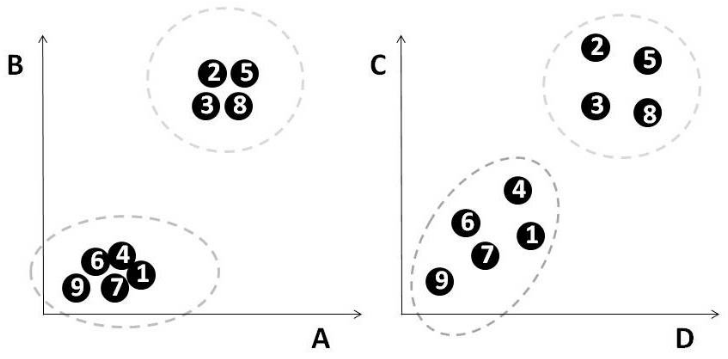


Figure 2.13: Subspace S_{AB} is dependent to S_{CD}, thus all objects are clustered in the same way. The mutual information between them is close to 1, even the variance and cluster shape may be different.

In general, dependent cluster means that the subspace clusterers generate dependent lower projections of a "true" cluster which is very distinctive to other clusters in a data set. Considering Figure 2.13, two fully dependent subspaces S_{AB} and S_{CD} are showed in a schematic view. Clearly, what makes them dependent is not the cluster shape or the variance, as they are (slightly) different in the two subspaces, but the same objects grouped in one subspace are clustered together in another subspace. Reducing dependence will greatly increase the accuracy and efficiency of any subspace clustering algorithm. And limiting the infinite search space to finite search space will greatly improve efficiency. In our approach this is easily achieved by automatically merging dependent subspaces with a greedy search method. Dependence is minimized in between the subspaces of the data set and we acquire almost independent subspaces by adopting ISA.

Now, clustering in these independent subspaces creates a novel type of cluster: independent cluster.

Definition 14. *(Independent Cluster) Two arbitrary clusters C_i and C_j $(i \neq j)$ are called independent clusters when they appear in two different independent subspaces S_1 and S_2. Independent Clusters differ maximally in their amount of information, i.e. their mutual information is near 0.*

$$C_i \subseteq S_1 \wedge C_j \neg \subseteq S_1 \tag{2.9}$$

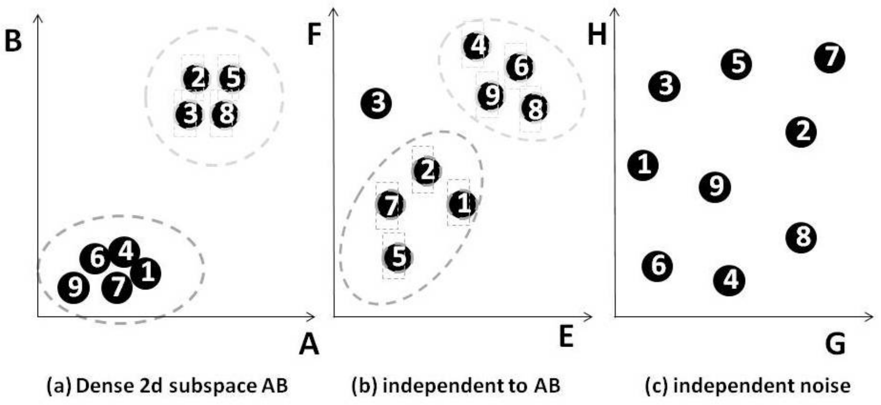


Figure 2.14: The clusters between subspaces S_{AB} and S_{CD} are independent as they co-exist in different independent subspaces. Even if the cluster shape is similar, the objects are grouped differently.

Clusters in different independent subspaces are highly independent from each other and can depict different structures hidden in the data set. Whereas clusters in the same subspace are dependent and more likely to reveal the distributions of objects in the data set. Consider the example in the schematic Figure 2.14 where three independent subspaces are shown, the independent subspaces S_{AB} and S_{EF} show clearly that the objects are clustered very differently, thus their included clusters are interpreted as independent clusters. There are no clusters in subspace S_{GH}, but S_{GH} is totally independent from subspace S_{AB} and S_{EF}.

Automation through Compression

The goals of generalized independent subspace clustering are two-folds: (1) searching for statistically independent subspaces and (2) achieving good clustering results in these found independent subspaces. The objective function of the first goal is as follows:

Definition 15. *(Objective function for searching for independent subspaces) Optimal results are acquired when the statistically independence between all pairs of subspaces is maximized, which is equivalent to the sum of mutual information I between all pairs of subspaces $I(S_i, S_j)$ $(i \neq j)$ being minimized:*

$$f = \min_{i \neq j} \sum_i \sum_j I(S_i, S_j) \tag{2.10}$$

In this paper, we use the Minimum Description Length Principle to reveal the relation between coding and model selection. Minimum Description Length (MDL) discriminates between competing models based on the complexity of each description by viewing statistical modeling as a means of generating descriptions of observed data [88]. Compressing a model M with MDL in general generates a flexible objective function balancing complexity between model M and data D.

$$L(D, M) = L(M) + L(D|M) \tag{2.11}$$

Given a set of models M, the best model is the one that minimizes equation 2.11where $L(M)$ is the length (in bits) of the description of one model in M, and $L(D|M)$ is the length (in bits) of the description of the data when encoded with M.

Compressing the Data Describing the data with a probability density function as close-to-real as possible raises another challenge. Thus, we use kernel density estimation (KDE) to estimate the probability density function of the data in each subspace. The definition of the multivariate kernel density estimation is as follows:

$$\widehat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h^d}\kappa(\frac{x-x_i}{h}) \tag{2.12}$$

where n denotes the number of objects, d stands for the dimension of the data set and κ denotes a d-dimensional, non-negative kernel function and $h = (h_1, \cdots, h_d)^T$ is a vector of bandwidths.

As a common choice in literature for KDE, we use the Gaussian kernel with mean 0 and variance 1. This leads to the following equation:

$$\widehat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n}\left(\prod_{j=1}^{d}\frac{1}{h_j}\kappa(\frac{x_j-x_{ij}}{h_j})\right) \tag{2.13}$$

where $\kappa(x) = e^{-x^2/2}/\sqrt{2\pi}$. The bandwidth h is selected according to [31]: $h_j = 0.9 \times n^{-1/(ds+4)} \times min(\sigma_j, IQR_j/1.34)$, where σ_j is the variance and IQR_j is the interquartile range of the j^{th} dimension.

To conclude the coding cost for the subspaces estimated by KDE, we need to encode the parameter costs for the KDE model. Here, only one parameter is of interest: the number of kernels λ, which is equals the dimensionality of the estimated subspace.

$$CC_p = \sum_{i=1}^{ns}\frac{\lambda_i}{2}\cdot log_2(n) \tag{2.14}$$

where ns is the number of subspaces.

Overall this leads to the following compression of the estimated independent subspaces group S: Let x_i be an object in the k^{th} subspace. The total coding cost of a data set is

$$L(D|M) = CC(S, ds) = \sum_{k=1}^{ns}\left(\sum_{i=1}^{n}\log_2\frac{1}{pdf(x_i)} + CC_{p,k}\right) \tag{2.15}$$

where $CC_{p,k}$ is the parameter cost of the k^{th} subspace, S is the group of the estimated independent subspaces and ds contains the dimensionality of each subspace.

Compressing the ISA Model. Compressing the ISA model using MDL has two challenges: first of all, data is unclustered and its distribution is unknown, therefore guessing its probability density function for model selection is quite hard. In addition, we use ISA as a black box, and different ISA implementations might need different parameter settings. In our approach, we compress parameters that are most likely demanded in every ISA model: the dimensionality ds of each subspace as well as the demixing matrix W.

The coding cost for the dimensionality of estimated independent subspaces is shown as follows:

$$CC = \sum_{i=1}^{ns} \left(\frac{d_i}{2} \cdot log_2 \frac{d}{d_i} \right) \tag{2.16}$$

where d denotes the dimension of the data set and d_i denotes the dimension of the i^{th} independent subspace.

Next, we encode the demixing matrix W which is a $d \times d$ square matrix. We encode each row of W by multivariate kernel density estimation as can be seen in the following equation:

$$CW = \sum_{i=1}^{d} \left(\log_2 \frac{1}{pdf(W_i)} + \frac{\lambda_i}{2} \cdot log_2(d) \right) \tag{2.17}$$

where W_i stands for the i^{th} row of W and λ_i is the number of bandwidth of the i^{th} row of W.

Finally, the overall coding cost of ISA model can be calculated as:

$$L(M) = CC + CW \tag{2.18}$$

Compressing the Clustering. As for clustering, in theory it would be possible to use any parameter-free, full-dimensional clustering method. However, this would not fit in generalized clustering to find arbitrary cluster shapes. For this reason, we adopt the soft clustering algorithm “Expected Maximization”(EM) algorithm [140] for clustering each independent subspace. The clustering process is automated by MDL, encoding the data with a multivariate gaussian distribution, as the EM algorithm adopts gaussian mixture models to cluster. Let $x \in R^d$ be a point of the i^{th} cluster c_i and $pdf_j(x)$ be a gaussian probability density function with the mean and variance of the j^{th} coordinate of the points in cluster c_i as parameters which are associated to c_i. The compression of a data point is

$$COP = \log_2 \frac{n}{|c_i|} + \sum_{j=1}^{d} \frac{1}{\log_2 (pdf_j(x_j))} \tag{2.19}$$

where $|c_i|$ stands for the number of objects in the i^{th} cluster.

The compression of a clustering is

$$COC = \sum_{k=1}^{K} \left(\sum_{i=1}^{|c_i|} COP_i \right) \tag{2.20}$$

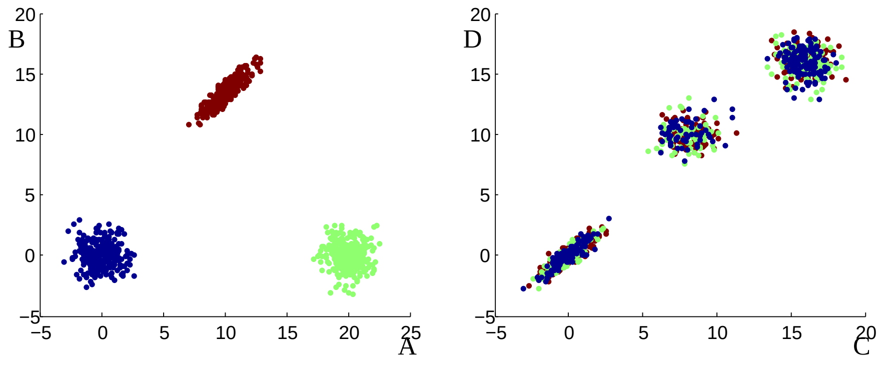
where K denotes the number of clusters in the data set.

Independence revealed by coding. Independence can be measured by mutual information, which accounts for the amount of information that one variable contains about another variable. There are so many kinds of estimators for mutual information that it is difficult to choose a proper one. Therefore, we use the method described in paragraph "Compressing the Data" to determine if two subspaces should be merged into one space or not according to their coding cost.

Example 2. *As example for showing independence by coding, consider four subspaces S_A, S_B, S_C and S_D. Their coding costs are* 4097.9*,* 3381.3*,* 3974.5 *and* 3951.3 *respectively. First, we compute the coding cost of every combination, i.e. S_{AB}, S_{AC}, S_{AD}, S_{BC}, S_{BD} and S_{CD} and then compute this result minus the sum of coding costs of each combination's components. The results are demonstrated in Table 2.3 showing possible independent subspace candidates. According to this results, subspace S_A and S_B should be merged, and subspace S_C and S_D should be merged as well, because their difference in coding costs are less than 0. Then, we evaluate if subspace S_{AB} and S_{CD} should be merged using the same routine. We compute their difference coding cost $S_{ABCD} - (S_{AB} + S_{CD})$ and the value is* 1213.4*. This encoding says subspace S_{AB} and S_{CD} should not be merged anymore. As end result, we acquire two independent subspaces S_{AB} and S_{CD} depicted in Figure 2.15.*

combination	S_{AB}-(S_A+S_B)	S_{AC}-(S_A+S_C)	S_{AD}-(S_A+S_D)
difference	**-600.26**	391.77	379.68
combination	S_{BC}-(S_B+S_C)	S_{BD}-(S_B+S_D)	S_{CD}-(S_C+S_D)
difference	364.29	352.26	**-1114.8**

Table 2.3: difference coding cost



(a) Subspace S_{AB} (b) Subspace S_{CD}

Figure 2.15: Two independent subspaces S_{AB} and S_{CD}

2.2.3 Algorithm

This section describes the algorithmic procedures of our algorithm ISAAC in detail. To gain information like the latent structure from arbitrary data, ISAAC in total works in three building blocks: Firstly, merging the possible subspaces depending on their independence. Secondly, applying automatic ISA to find the best partition for independent subspaces inside the data. And finally, clustering data in every estimated independent subspaces.

Mechanism of ISAAC

Compressing the search space. Limiting the search space in a heuristic way is crucial for every subspace clustering algorithm. For example, if the dimension of a original data set is d, the number of possible axis-parallel subspaces is 2^d, while the number of arbitrarily oriented subspaces is infinite. Besides, we need some optimal suggestions for parametrizations in the next processing step, which is the automatic independent subspace analysis, so that our algorithm does not run infinitely.

To effectively search for a promising partition, we develop a greedy search algorithm which balances exploration and complexity. We firstly partition the attributes into single dimensional subspaces and compare the compression cost of a single subspace to that of a merged one according to the subspace data compression costs for ISA as defined in equation 2.12.

```
Algorithm 1: Automatic greedy search procedure of ISAAC
Input: $d$ dimensional data $X$
Output: estimated independent subspace set $S$ and $W$
// initialization;
$ds$(1:d) $\leftarrow$ 1, $[S, W] \leftarrow ISA(X, ds)$;
$L(0) \leftarrow L(D, M)$ with input parameters $S$, $W$, $ds$;
$iteration \leftarrow 1$, $cnt \leftarrow 1$, $minC \leftarrow 1$;
// Compressing the search space;
while $length(ds) > 1$ do
    for $i \leftarrow 1$ to $d$ do
        for $j \leftarrow i + 1$ to $d$ do
            $t \leftarrow CC$([S$_i$;$S_j$],[ds($i$)+ds($j$)])-
            CC([$S_i$;$S_j$],[ds($i$),ds($j$)])
            if $t < 0$ then
                put $i$, $j$, and $t$ in the $cnt^{th}$ column of matrix
                $codingMatrix$;
                $cnt \leftarrow cnt + 1$;
            end
            if $t \leq minC$ then
                $minC \leftarrow t$;
            end
        end
    end
    if $minC > 0$ then
        break;
    end
    $[C, index] \leftarrow sort(codingMatrix(1, :))$;
    $codingMatrix1 \leftarrow codingMatrix(2:3, index)$;
    merge subspaces which labels are stored in
    $codingMatrix1(1, 1)$ and $codingMatrix1(2, 1)$;
    for $i \leftarrow 2$ to $cnt - 1$ do
        if subspaces which labels are stored in
        $codingMatrix1(1, i)$ and $codingMatrix1(2, i)$ have not
        merged then
            merge these two subspaces;
        end
    end
    acquire a new $ds$;
    $[S, W] \leftarrow ISA(X, ds)$;
    $L(iteration) \leftarrow L(D, M)$ with input parameters $S$, $W$, $ds$;
    $iteration \leftarrow iteration + 1$;
end
sort $L$ and output the best $W$, $S$ and $ds$;
```

Figure 2.16: Automatic greedy search procedure of ISAAC

If the compression cost of a merged subspace is less than the summed up compression cost of its two components, we put the labels of these two subspaces and their difference in compression costs into a list for ranking. After ranking, we merge two subspaces according to the ranked difference in compression cost ascending. At the beginning of each merging process, we check if the specific two subspaces have been merged. If they have not merged, we merge them and the merging process continues until traversing the ranked list. Then we acquire the parameters for the next step in ISA. After processing the next building block we obtain a new group of estimated independent subspaces S and the demixing matrix W. This procedure continues until convergence.

Automatic Independent Subspace Analysis After finding an optimal candidate for parametrization in the first building block, we run ISA with these found parameters for generating the best independent subspaces. ISAAC in theory works with every ISA implementation as it uses ISA as a black box. For this ISAAC implementation we adopted the ISA proposed in [180], which solves an to ISA related problem the best in our experiments. The ISAAC in [203] uses a different version of ISA which is a combination of standard ICA [39] and merging the independent components. As result of this second building block, we gain the *dimensionality of each found independent subspace* d and the transformed data . The pseudocode of these two building blocks is given in Algorithm 1.

Independent Generalized Clustering This last building block focuses on the actual clustering process of ISAAC in each prior found independent subspace. All necessary input parameters for this clustering procedure, which are the (estimated) independent subspaces S and their dimensions d, are automatically found by the above two building blocks. Note that every single subspace to be processed separately can also be dealt with in parallel, which makes our approach more efficient if the data set is very large. Clearly, the found clusters in one subspace are independent to clusters found in other subspaces due to the maximized independence relation between these subspaces. For automatically choosing the proper number of clusters in each subspace, we apply MDL, i.e. equation 2.20, to the EM algorithm that enables us to run the EM algorithm without previous parametrization of the number of clusters K. A Gaussian distribution with mean and variance of the i^{th} coordinate of a cluster as parameters is used here to describe the probability density function of each cluster. The clustering procedure of ISAAC can be found in Algorithm 2.

```
Algorithm 2: Clustering procedure of ISAAC
Input: $d$ dimensional estimated independent subspace group $S$ and
       their dimensionalities $ds$
Output: the cluster labels in each subspaces and the overall quality
        evaluation value
// input the best partition to EM;
$K \leftarrow 1$;
while true do
    $clustering \leftarrow EM(S, K, ds)$;
    $codingCost(K) \leftarrow COC(clustering)$;
    $K \leftarrow K + 1$;
    // find the best number of clusters;
    if $K \geq 2$ and $codingCost(K-1) < codingCost(K)$ then
        break;
    end
end
evaluate the best clustering by F1 measure;
```

Figure 2.17: Clustering procedure of ISAAC

Complexity Analysis of ISAAC

The runtime complexity of whitening the data, a first step in ISA, is $O(nd^2)$. Here, n is the number of objects and d is the number of dimensions in a data set. Since ISAAC is based on ISA proposed in [180], we need to check on the runtime of fastICA [101](the underlying method for finding independent components in ISA) and the grouping process done with mutual information. The computational cost per iteration of fastICA is $2n(d+1)$. If the iteration number of fastICA is m, then the overall runtime complexity of fastICA is $O(2mn(d+1))$. The next step in ISA is the actual clustering step. Since the cost of computing pairwise mutual information is $O(n)$, the time cost of clustering independent components is $O(nd^2)$. Thus, the total runtime complexity of ISA is $O\left((d^2+2md+2m)n\right)$. For MDL the runtime complexity of compressing the estimated independent subspaces and the demixing matrix is $O(nd+d^2)$. Thus, the total time complexity of finding the best partition is $O\left((d^2+(2m+n)d+2m)n+d^2\right)$. Since m is a constant, the time complexity becomes $O(d^2n+2mnd+n^2d)$. For the clustering process, the runtime of determining the number of clusters is $O(Knd)$, where K is the final number of clusters determined by MDL. Therefore, the total runtime complexity of ISAAC is $O(d^2n+mnd+n^2d)$ plus the computational complexity of the EM algorithm $O(Kni)$, where i is the number of iterations of the EM algorithm.

2.2.4 Experimental Evaluation

In this section, we compare ISAAC with the non-redundant but axis-parallel subspace clustering methods INSCY [18], STATPC [139] and RESCU [144]. Together with the arbitrary-oriented subspace clusterers ORTH1, ORTH2 (*orthogonal clustering*, and *clustering in orthogonal subspaces*, both presented in [51]) and MSC [153], we have several very related competitor systems. All comparison methods are implemented in Java, the axis-parallel methods implementation is from in the OutRules framework [146], a WEKA extension. ISAAC is written in Matlab, but a Java version exists as well [203]. The parameters for all subspace clustering methods are set according to their original chapters. For MSC, we provide the true number of subspaces and clusters. In real-word data, we set the number of subspaces for MSC equal to those found by ISAAC. For ORTH1 and ORTH2, we provide the true number of clusters. All runtime experiments were done on the same machine with an Intel Core Quad i7-3770 with 3.4 GHz and 32 GB RAM. For quality evaluation we report the F1 [6] value for our synthetic data and real data sets. The F1 value of a clustering is the harmonic mean of its precision and recall and most frequently reported in state-of-the-art research of subspace clustering [145].

Synthetic Data

The generation process of our synthetic dataset “sync” is as follows: we assume we have τ two-dimensional “ground truth” subspaces with a clustering in each. Half of these subspaces contain four clusters; the remaining subspaces contain six (varying size of clusterings). Correlations between the observations in each cluster are obtained by 1) starting with n observations generated from uncorrelated standard normal variables, 2) choosing a correlation matrix C and scalar $r \in [0, 1]$ in such way that $c_{i,i} = 1$ and $c_{i,j} = r, i \neq j$, and 3) applying C's Cholesky transformation to the observations. Clusters are then positioned in the two-dimensional space by their respective centers – each a random sample from the set $[20, 80]^2$. To simulate non-redundancy between clusterings, we randomly permute the object IDs in each subspace before merging them to a full-space dataset.

Quality evaluation

For the evaluation of the synthetic data set *sync*, we vary the number of hidden independent subspaces τ from 1 to 10. $\tau = 10$ means *sync* is composed of 10 subspaces, each of which has the dimension size of 1, while $\tau = 1$ means *sync* only contains one 10-dimensional subspace.

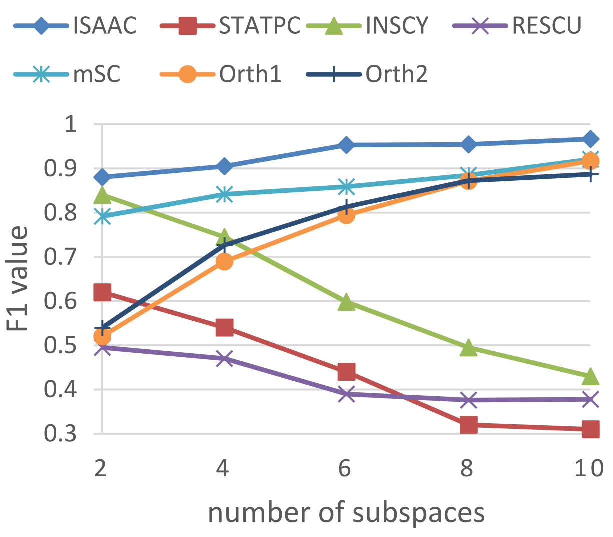


Figure 2.18: F1 value of synthetic data set *sync*

Figure 2.18 shows the effect of increasing the number of subspaces. Clearly, ISAAC outperforms all evaluated competitor systems in quality and even continually increases the quality the more hidden independent subspaces in τ are added. This is specifically good as ISAAC is not “helped’ ’with the provision of any input parameters (the other multiple-clustering techniques require the correct number of subspaces and/or clusters). Besides, we can see the trend, that all axis-parallel competitors can not deal at all with independent subspaces and their quality decreases rapidly, while the quality of all arbitrary multi-view clusterers increases continuously from relative low starting points when $\tau = 2$.

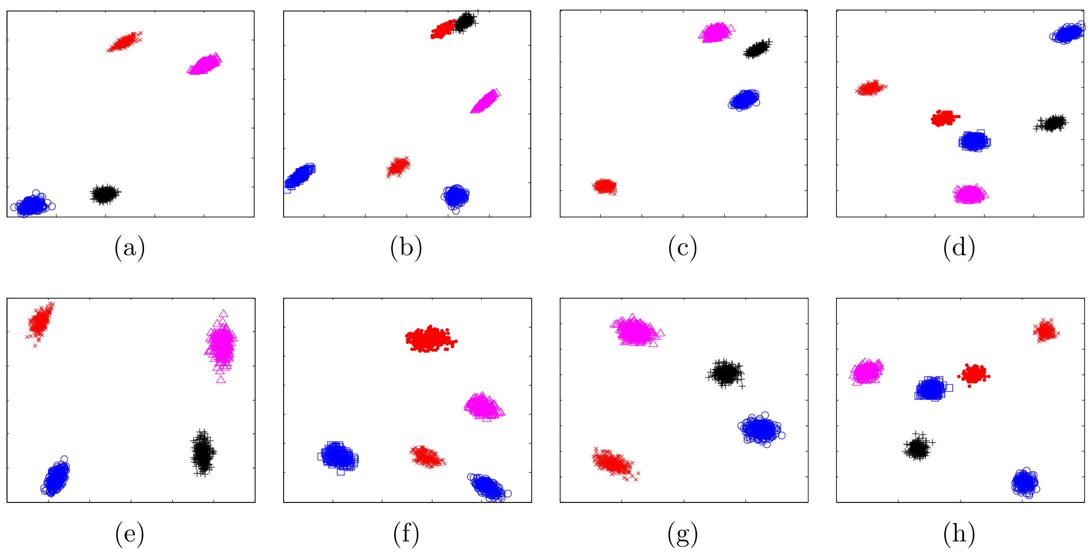


Figure 2.19: Cluster Quality. the first row: synthetic data set "sync" with four two-dimensional independent subspaces; the second row: the four independent subspaces and clusterings found by ISAAC.

Besides F1-measurement, there is another important attribute of ISAAC regarding the cluster shapes. When at least two subspaces exist, ISAAC can always discover all hidden independent subspaces and clusters in each subspace efficiently, but the shape of some clusters may be changed due to space transformations after running ISA, i.e. each attribute in the transformed space is a linear combination of all attributes in the original space. This is depicted in Figure 2.19. The first row of Figure 2.19 shows the four independent subspaces contained in "sync". The second row of Figure 2.19 demonstrates the four independent subspaces (corresponding to the subspaces depicted in the first row) and clusterings found by ISAAC. Compared to the original subspaces, we can see from the figure that the found subspaces are rotated because ISA tries to find arbitrarily-oriented subspaces which are a linear combination of the original ones. When checking the found subspace (the second row, second column of Figure 2.19, which corresponds to the subspace shown in the first row, second column of Figure 2.19), we find the top two very-close clusters are not separated by ISAAC. In other independent subspaces, clusters are separated cleanly.

Runtime evaluation

Although we mainly focus on clustering quality, we briefly evaluate ISAAC's runtime efficiency as well.

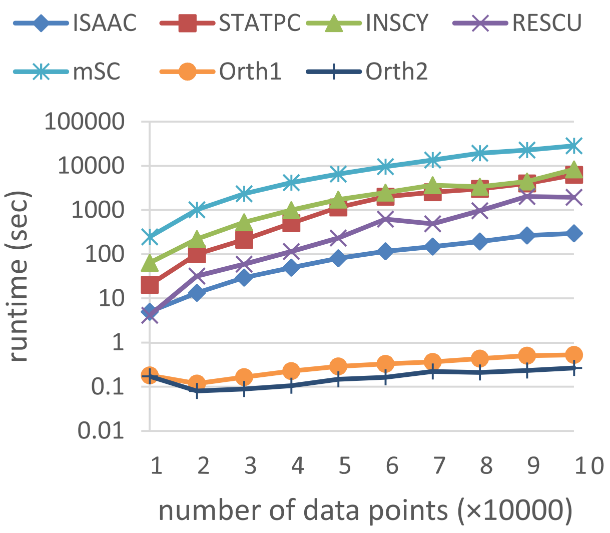


Figure 2.20: Runtime evaluation on different database size on sync

Figure 2.20 shows the results for varying the database size n from [10k, 100k]. Clearly, ISAAC outperforms all methods except ORTH2 and ORTH1in runtime. The orthogonal subspace clusterers dominance can easily be explained by strict parameter settings that need no costly heuristic search to find the candidate subspaces as ISAACs automatic approach needs. At this point it is important to note that ISAAC is the only fully-automatic method being evaluated: it invests time to search for appropriate model-values in various stages of the framework (this additional effort is included in the experimental results). Despite this – and ignoring constant factors – ISAAC "holds its own" in terms of the run-time growth rate. Its observed quadratic growth rate in n is comparable to MSC, INSCY and STATPC (ORTH2 behaves linearly, and RESCU's runtime grows with n^3).

Real Data

In this section, we evaluate the performance of ISAAC and the six comparison methods on nine real world data sets. The parameters of the comparison methods are set according to their original papers.

Quality Measurement

For quality evaluation, we compare the F1 measure of nine real-world datasets. The Breast (Wisconsin Diagnostic), Ecoli, Spam, Shuttle, Musk and Connectionist Bench (Sonar, Mines vs. Rocks) datasets are from the benchmark UCI repository (`http://archive.ics.uci.edu/ml/datasets.html`). The metabolic dataset is from a PKU newborn screening [128]. Dancing Stick Figures (DSF) [83] is a multi-view dataset with 900 samples of 20×20 images across nine stick figures (Figure 2.21). Amsterdam Library of Object Images (ALOI) [74] collection consists of images of 1000 common objects taken from various angles and under various illumination conditions. We chose four different objects (Figure 2.22) with all their images taken from different viewing directions. We extracted color and texture features with 611 dimensions for each image using the method proposed in [15] (code can be found here[6]). Then for DSF and ALOI, we further apply PCA as a preprocessing step (also used in [188, 83]), retaining at least 90% of the variance (five principal components).

The results of all algorithms on real data sets in terms of F1 value found by ISAAC are listed in Table 2.4. Clearly ISAAC outperforms the other algorithms on most of these data sets regarding the F1 value.

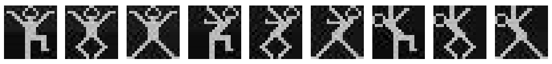


Figure 2.21: Nine raw samples from the Dancing Stick Figures.

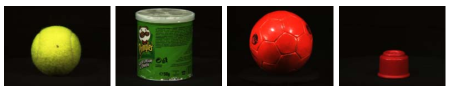


Figure 2.22: Four objects of different shapes (ball and cylinder) and colors (green and red) from ALOI.

ISAAC is deployed in our proposed automated fashion (parameter-free). ORTH1 and ORTH2 require the number of clusters, so we use the number of class labels in each respective dataset. For DSF we inform ORTH1 and ORTH2 that there are three clusters in each subspace (based on the qualitative intuition in the next section). In addition to the number of clusters, MSC requires the number of subspaces – here we provide it with the same value found by ISAAC in all cases.

[6]http://www.cat.uab.cat/Research/ColorTextureDescriptors/

Table 2.4: F1 measure on real-world data (with dimensions (n;m)). $*$ failed because of non-trivial bugs in the OpenSubspace implementation [145].

Dataset	Metabolic (709;10)	Ecoli (336;7)	Breast (569;30)	Spam (4601;57)	Shuttle (43500;9)	C. Bench (208;60)	Musk (476;166)	DSF (900;5)	ALOI (288;5)
ISAAC	**0.81**	**0.71**	**0.71**	**0.69**	**0.78**	**0.66**	**0.67**	**0.86**	**0.87**
mSC	0.41	0.50	0.68	0.68	0.65	0.61	0.65	0.74	0.79
Orth1	0.54	0.43	0.69	0.68	0.63	0.60	0.65	0.71	0.67
Orth2	0.54	0.42	0.69	0.68	0.65	0.52	0.65	0.74	0.62
STATPC	0.44	0.32	0.61	0.68	0.19	0	0	0.60	0.42
INSCY	0.29	0.11	0.65	0.01	–*	0	–*	0.62	0.27
RESCU	0.26	0.07	0.39	–*	–*	0	–*	0.58	0.33

Table 2.4 reports the F1 measure for each dataset and algorithm. We see that ISAAC obtains a stronger F1 measure in all cases, even outperforming techniques like mSC, Orth1 and Orth2 which have the advantage of being given the correct number of clusters as a parameter.

In the following we will only interpret the results of ISAAC on breast and metabolic data sets in detail because ISAAC found some two or three dimensional subspaces on these two data sets. We will also qualitatively interpret and compare the results for Dancing Stick Figures and Amsterdam Library of Object Images multi-view datasets. For the other datasets, since we do not have "ground truth" subspaces, we omit their interpretation.

Breast Cancer Wisconsin (Diagnostic) Data Set The Breast Cancer Wisconsin (Diagnostic) data set deriving from a study on breast cancer consists of 569 instances which are labeled to two classes malignant (M: 212 instances) and benign (B: 357 instances). Each instance is described by 30 numerical attributes. ISAAC found 4 subspaces with dimensions size of 15, 6, 6, and 3 respectively. We depict the subspace of dimension size 3 in Figure 2.23. In this subspace, ISAAC detected five clusters, which had 418, 118, 27, 3 and 3 instances in each cluster. 31 malign instances and 2 benign instances were clustered as noise. 63 malign instances were wrongly clustered as benign instances. Every axis in Figure 2.23 corresponds to one independent component that is a linear combination of all original attributes with different weights.

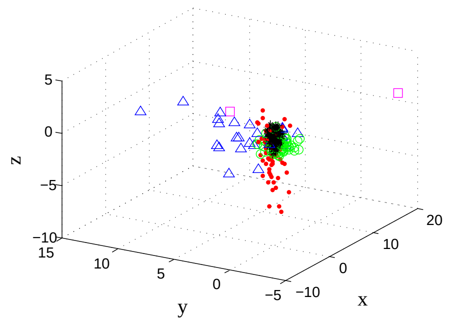


Figure 2.23: ISAAC on breast cancer Wisconsin (diagnostic) data set

The weights of attributes which make up every axis in the 3 dimensional subspace are demonstrated in Figure 2.24. From Figure 2.24 we can see which attribute has a relatively higher contribution to the axes of the subspace. For example, attributes with id number 8, 18, 20 and 28 have a higher contribution to z-axis.

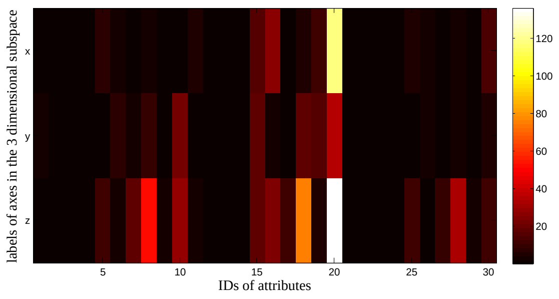


Figure 2.24: The weights of attributes making up axes in the 3 dimensional subspace (The weights of attributes are represented by colors)

Metabolic Data Set The metabolic data set originates from a screening program for metabolic disorders in newborns. All instances are described by 10 attributes representing metabolite concentrations and can be clustered to 5 classes with size 100, 101, 51, 51, 197 respectively. ISAAC detected 4 independent subspaces with the dimension size of 16, 4, 4 , 2 each. Three 2-dimensional independent subspace is depicted in Figure 2.25.

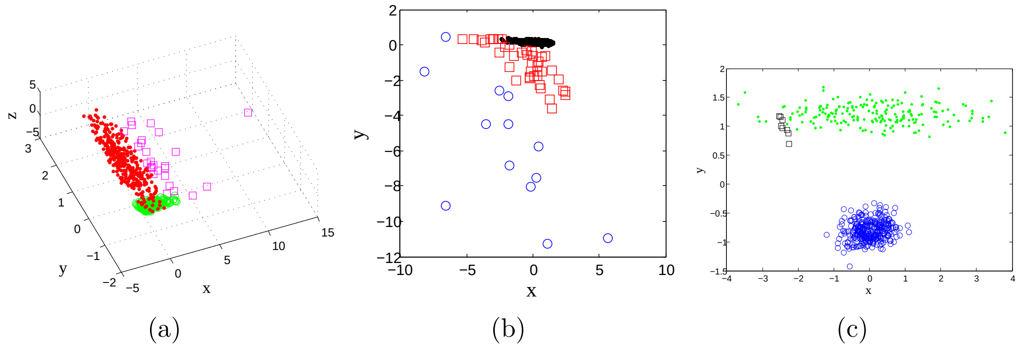


Figure 2.25: ISAAC on metabolic data set - three independent subspaces

Every axis in the figure corresponds to one independent component that is a linear combination of all original attributes with different weights. Figure 2.25(a) shows a different independent subspace from Figure 2.25(b) and (c), which means their contained clusters are also independent. The independence is given by the way their objects are clustered. The subspace (a) and (b) each contains 1 cluster, and the subspace (c) contains 3 clusters. In subspace (c), the blue cluster contains 299 objects, the green cluster contains 193 objects and the black one contains 8 objects. Interpreting the results is relatively easy when knowing how the weights are distributed in every subspace. The weights of attributes which make up every axis in every independent subspace are demonstrated in Figure 2.26. We can see from Figure 2.26 which attribute has a relatively higher contribution to the axes of every independent subspaces. For example in subspace (c), attributes with ID number 4, 7 and 10 contribute more to the x-axis.

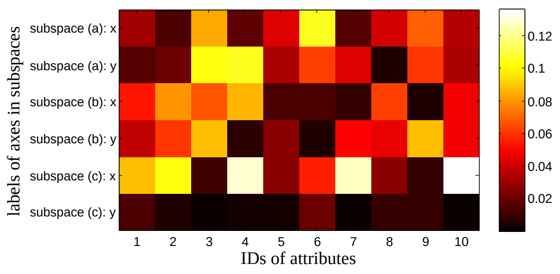


Figure 2.26: The weights of attributes making up axes in 3 independent subspaces (The weights of attributes are represented by colors)

Dancing Stick Figures Dataset In the DSF data, ISAAC finds three independent subspaces. The first and second subspace contain three clusters and the third contains four. Figure 2.27 depicts the means of the detected clusters in the first and second subspaces (we don't show clusters in the third subspace because they are not very interpretable). We clearly see a compelling separation into upper- and lower-body motions (two non-redundant views on the data). In comparison, we see in Figure 2.27 the two subspaces found by ORTH2 (the best of the competition from Table 2.4). Here ORTH2 fails to detect any intuitive and convincing perspectives.

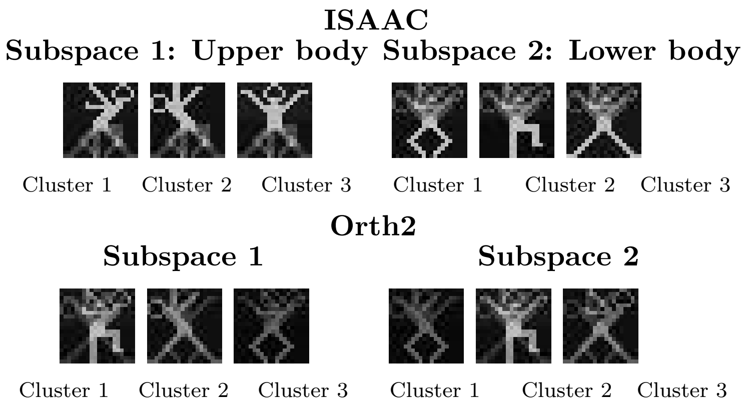


Figure 2.27: The means of the clusters detected by ISAAC (top) and ORTH2 (bottom) in two subspaces (Dancing Stick Figures data). ISAAC identifies clear upper- and lower-body perspectives.

Amsterdam Library of Object Images Dataset For the ALOI data, ISAAC finds three independent subspaces. The means of the detected clusters in the subspaces are depicted in Figure 2.28. Again the subspaces show three interesting perspectives on the data: one groups by *shape* (cylinder and ball), another by *color* (red and green) and the other by *size* (small and big). It is very interesting that ISAAC detects the subspace in which objects of similar sizes are clustered. In comparison, the two subspaces detected by MSC (the best of the competition from Table 2.4) show that it fails to identify the separation between color and shape.

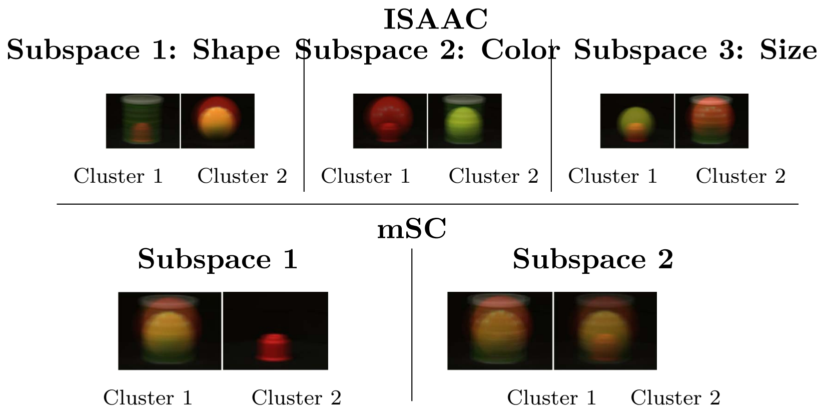


Figure 2.28: The means of the clusters detected in the ALOI data by ISAAC (three subspaces, top) and mSC (two subspaces, bottom). ISAAC successfully identifies subspaces which partition color and shape. In addition, it finds subspace in which objects of similar sizes are grouped together. (*best viewed in color*)

2.2.5 Related Work And Discussion

Subspace clustering algorithms in axis-parallel subspaces Subspace clustering is divided in two categories, i.e., axis-parallel subspace clustering and arbitrarily oriented subspaces clustering. The first category can be divided to the grid-based subspace clustering and the redundancy-reducing-based subspace clustering. The grid-based subspace clustering algorithms include CLIQUE [11], ENCLUS [44], MAFIA [147], etc., which adopt a global density threshold in a bottom-up way to search for clusters. The resolution of the grid is very important in the performance of these grid-based subspace clustering algorithms. With inappropriate grid resolution, arbitrary oriented or shaped clusters may not be discovered. SCHISM [174] extends these grid-based subspaces clustering algorithm by using a variable threshold to detect dense regions, but when the search subspace become larger, the function of the variable threshold that is equal to that of the global density threshold. The redundancy-reducing-based subspace clustering algorithms including INSCY [18], STATPC [139], RESCU [144] are also strongly related to our approach thus by finding independent subspaces, non-redundancy is given all the time. INSCY combines in-process redundancy pruning with novel index structure, the SCY-tree, for efficient subspace clustering. STATPC selects a suitable set $R^{Reduced}$ from R, represented by a reduced, non-redundant set of (axis-parallel) statistically significant regions that in a statistically meaningful sense to search efficiently. RESCU presents a global optimization which detects the most interesting non-redundant subspace clusters by taking a global look at overlapping clusters. However, the algorithms listed above cannot detect clusters which are situated in arbitrarily oriented subspaces.

Subspace clustering algorithms in arbitrarily oriented subspaces Arbitrarily oriented subspaces clustering can also be called as generalized subspace clustering, or correlation clustering. The fundamental technique utilized by most approaches is PCA. ORCLUS [8] is the first method of arbitrarily oriented subspace clustering, which combines PCA with kmeans clustering. ORCLUS picks seeds at first, and includes three procedures: Assign, FindVectors and Merge. At the beginning of the “Assign procedure”, it partitions the database into current clusters by assigning each point to its closest seed. At the end of this procedure, each seed is replaced by the centroid of the cluster which was just created. In “FindVectors” procedure, the dimensionality of the subspace for each current cluster is found. In the “ Merge procedure”, a measure for testing the suitability of merging two clusters by examining the projected energy of the union of the two clusters in the corresponding least spread subspace is designed. The algorithm terminates when the merging process over all the iterations has reduced the number of clusters to k. 4C [30] combines PCA with DBSCAN to search for arbitrary linear correlations of fixed dimensionality. Instead of PCA, CASH [3] utilizes the Hough transform to find arbitrarily oriented subspace clusters by mapping the data space to a parameter space to detect correlations within the attributes. COPAC [5] assigns a local correlation dimensionality to each object and then partitions these objects according to their local correlation dimensionality. ERiC [4] partitions the objects of the database according to their correlation dimensionality, then the points within each partition are clustered using a flat correlation clustering algorithm followed by a bottom-up strategy to explore the relationships among the correlation clusters. However, the methods listed above either need many input parameters to determine the number of detected correlation clusters or can only detect correlation clusters with less noise, which restricts their broad applications.

Multiple Clustering is also a related field for our method. Multiple clustering seeks to partition a given set of objects in different ways, which represents different perspectives of the data. COALA [20] generates multiple clusterings by using instance level constrains. NACI [52] is driven by using mutual information to optimize the dual objective functions of both quality and dissimilarity. [188] presents a computationally efficient nonparametric entropy estimator to quantify both clustering quality and distinctiveness. However, the above methods can only generate two alternative clusterings. MVGen [84] generates multiple clusterings of data by using multiple mixture models. MVGen uses the iterated conditional modes (ICM) principle and adopts Bayesian model selection to make a balance between the complexity of the model and its goodness of fit. SMVC [83] integrates semi-supervised clustering with multiple clustering and uses variational Bayesian methods for efficient learning. However, the purposes of MVGen and SMVC are to detect multiple, overlapping clustering views which are not non-redundant. Multiple Stable Clustering [94] detects multiple stable clusterings using the idea of clustering stability based on Laplacian Eigengap. But the found multiple stable clusterings cannot guarantee diversity, i.e., some clusterings are redundant and potentially difficult to interpret. mSC [153] integrates the relaxed spectral clustering objective with the Hilbert-Schmidt independence criterion (HSIC) to find multiple non-redundant views, and then uses spectral clustering to find clusters in each view. Orthogonal projection clustering (OPC) [51] uses two strategies, (1) orthogonal clustering (Orth1), and (2) clustering in orthogonal subspaces (Orth2), to partition data to achieve multiple clusterings. The last three non-redundant multiple clustering algorithms achieve the same goal as ISAAC. Differing from Orth1 which directly seeks non-redundant clusterings, ISAAC indirectly seeks multiple non-redundant clusterings by using ISA to generate independent subspaces. Thus, clusterings in those subspaces are independent (non-redundant). The strategy is very similar to those of Orth2 and mSC which also firstly seek independent or orthogonal subspaces followed by clustering in those subspaces.

2.3 Conclusion

In this chapter, we introduced two novel approaches to subspace clustering: One, NORD, that balances the quality of a clustering with the novel information gained and the other, ISAAC, a parameter-free technique for generalized subspace clustering. ISAAC can find multiple clusterings in arbitrarily-oriented subspaces of heterogeneous dimensionality such that pairwise clusterings are highly statistically-independent (non-redundant) and contain potentially-differing numbers of clusters while NORD works on axis-parallel clusterings only. Both algorithms can be considered parameter-free in the sense, that no sensitive input parameter are necessary to gain a highly valuable result. This automation is achieved through the MDL principle, where model parameters are selected by balancing accuracy and complexity. An efficient, MDL-driven greedy search heuristic helps ISAAC and NORD to find the best space partition. As the first information-theoretic algorithms that are applied to the topic of non-redundant subspace clustering, we showed clearly in our experiments that the heuristic search method is relatively fast compared to other state-of-the-art algorithms and achieves a high quality even without the need for parameters. Last but not least we proposed a visualization of the clustering results of NORD, that intuitively shows the relationship between quality and novelty. To conclude, we feel that our proposed solutions are able to yield more useful subspace clustering results: Instead of yielding a potentially overwhelmingly large set of high-quality clusters, that might be highly redundant, our solutions narrows down the space of interesting clusters to the most representative clusters.

CHAPTER 3

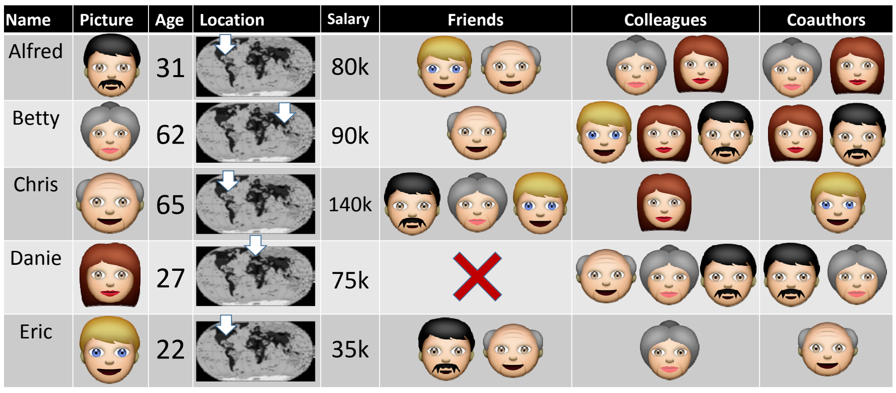
Methods for Clustering Graph Data

Parts of this chapter have previously been published in ICDM 2013 [68] [97]

In this chapter we focus on undirected graph data in social networks like Facebook, Twitter, IMDB and DBLP. While the heterogeneous graph model seems to estimate reality the best [87], a complex model has methodological a high overhead. We chose an easier model for our two automatized community detection methods. The first method, IROC, finds overlapping communities and reduces their redundancy automatically for attributed graphs like the two methods for vector data in the chapter before. The second method, Cxprime, finds communities in very sparse graphs and can detect whether the community itself is sparse or dense.

3.1 IROC - Non-redundant Overlapping Clustering

In recent years, not only the volume, but also the complexity of data has increased significantly. Objects are associated with textual information, multi-media data, social interaction data, and other types of information. This trend is facilitated by two major factors. First, current trends in technology such as smart phones allow individual users to easily gather a huge variety of data on-the-fly. Second, there is a new user mentality of utilizing this technology to voluntarily publish and share their gathered information publicly in social networking sites, blogs, wikis

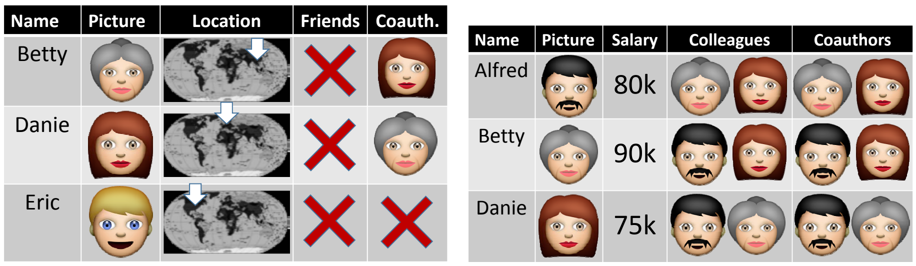


Name	Picture	Age	Location	Salary	Friends	Colleagues	Coauthors
Alfred		31		80k			
Betty		62		90k			
Chris		65		140k			
Danie		27		75k	✗		
Eric		22		35k			

Figure 3.1: Attributes and Social Networks of Users

and other platforms. Consequently, entities are described by a potentially large set of attributes, such as attributes of users, and potentially many networks describing their interactions, such as friendships, collaborations and spatial proximity. Such a complex network can be captured by a multi-mode attributed graph. This multi-mode attributed graph connects two aspects of information: First, the structural information described by multiple types of interactions of the network. Second, the attributes of each node. Combining both aspects of information gives the chance to answer questions like “What does a circle of friends have in common?” or “what do people in my work environment like and what joins them together?”. Mining answers for these questions does not only unveil social aspects of groups, but also opens a new area of targeted marketing, where offers can be tailored towards groups of people, rather than individual recommendations only. Questions like this can be answered to some extend by just clustering attributed graphs, as existing algorithms propose in [211], [196] and [210]. However, these works have in common that they mine a non-overlapping clustering, where each database entity may be part of at most one cluster.

Example 3. *An example of a database setting where objects are interacting in different layers is shown in Figure 3.1. Here, each data record corresponds to a user, represented by a number of attributes stored in the database, including their name, age, spatial location and the user’s salary. In addition, we are giving information about social networks that the user participates in, including a friendship network, a citation network and a collaboration network. Clearly, given all this information about users, it becomes more challenging to identify clusters of users. Here, we can see that the users Alfred, Betty and Eric form a clique in both their colleague as*



(a) Low Quality Cluster (b) High Quality Cluster

Figure 3.2: Clusters in the Multi-Model Network of Figure 3.1

well as their co-author networks. At the same time, these three user are entirely disconnected in the friends network. In contrast, users Alfred, Chris and Danie are forming a clique in the friendship network, and are located in the same spatial region. Arguably, we can already identify two clusters: one cluster covering objects Alfred, Betty and Eric in the modes collaboration *and* citation*, and one cluster covering users Alfred, Chris and Danie in modes* friends *and* location*.*

In our example of Figure 3.1, user Alfred is contained in both clusters that we identified in Example 3. Without the notion of overlapping clusters, we are forced to assign Alfred to either his circle of private friends, or to his circle of work friends, thus incurring a significant loss of information. In practice, a single entity may be part of a very large number of communities, such that the enforcement of non-overlapping clusters yields an immense loss of information.

Summarizing, this toy example has shown the potential information preservation by allowing the overlap of clusters. However, in practice, two overlapping clusters can be highly redundant, sharing nearly the same set of entities, attributes and networks. Such overlapping will incur *information redundancy*, which implies that one cluster contains significantly less information when given information about the other cluster. In this chapter, our proposed algorithm aims to detect overlapping clusters while avoiding information redundancy using an information theoretical approach. The advantages of the proposed algorithm are:

- **Discovery of overlapping communities**: Our method allows to discover overlapping clusters combined in attribute and network space.
- **Finding coherent attribute subspaces**: Our approach automatically finds a corresponding subspaces of modes and attributes of a cluster, allowing overlapping subspaces.

- **Redundancy Minimization**: Using Minimum Description Length (MDL) principle [170] to measure the quality of a clustering, we can obtain a balance between quality and redundancy of a clustering.
- **Automation**: Our approach does not require the user to specify any data specific parameters.

The remainder of this chapter is organized as follows: In the following section, Section 3.1.1, we formally define the problem of clustering multi-mode attributed graphs. In Section 3.1.2, we elaborate our coding scheme, which is necessary to avoid parametrization and balances among quality and redundancy. Exploiting this coding scheme, we present our algorithm for **I**nformation Theoretic non-**R**edundant **O**verlapping **C**lustering (**IROC**) in Section 3.1.3. Our experiments in Section 3.1.5 show that *IROC* achieves high clustering quality for existing community-labelled attributed social networks, and our proposed improvement using tensor factorization, *TF-IROC* fills this gap by sacrificing some of *IROC*s effectivity in order to achieve scalability to very large networks. In Section 3.1.6 surveys the related work.

3.1.1 Problem Definition

This section describes the notation used throughout the chapter.

Definition 16 (Multi-Mode Attributed Graph). *A M-mode T-attributed graph is a triple $\mathcal{G} = (V, E, \Lambda)$, such that $V = \{v_1, v_2, ..., v_N\}$ is a set of vertices, $E \subseteq V \times V \times M$ is a set of edges, and $\Lambda = \{\lambda_1, ..., \lambda_T\}$ is a set of attribute functions $\lambda_i : V \mapsto \theta_i$ each mapping vertices to an attribute space θ_i. Further, we let A^k denote the $|V| \times |V|$ adjacency matrix of the network corresponding to mode k having $A^k_{ij} = 1$ if $(v_i, v_j, k) \in E$ and $A^k_{ij} = 0$ otherwise; and we let F denote the $|V| \times T$ attribute matrix having $F_{ij} = \lambda_j(i)$.*

Example 4. *As an example, consider Figure 3.1, which shows a $M = 3$-mode $T = 3$-attributed graph having five vertices $V = \{Alfred, Betty, Chris, Denie, Eric\}$ which correspond to individual users. Each of the three modes {Friends , Colleagues, Co-authors} corresponds to different social graph, describing social connections between users. Here, each of the three social graphs A^1, A^2 and A^3 is depicted by an adjacency list using pictures of the five users. The set of edges*

E contains, for instance, the edge $(Alfred, Chris, 1)$, implying that users Alfred and Chris are connected in mode 1 which corresponds to the friends *graph. The three attributes of users correspond to age, geo-location, and salary of corresponding users. For instance, the attribute function λ_1 maps each user to their age, such as $\lambda_1(Alfred) = 31$.*

In this subchapter, we aim at mining knowledge from the attributed graph by detecting non-redundant overlapping clusters. As attributed graphs possess both structural and attribute information, the clustering of this type of data covers structural and attribute information. A cluster C is a subset of a multi-mode attributed graph G. Specifically, a cluster is defined by a set of vertices $V' \in V$ and a set of attributes $\Lambda' \in \Lambda$.

In the following, we formally define a multi-mode attributed graph cluster as a special case of a multi-mode attributed subgraph.

Definition 17 (Multi-Mode Attributed Subgraph)**.** *Let $\mathcal{G} = (V, E, \Lambda)$ be a M-mode T-attributed graph. A subgraph of $\mathcal{G}$ is a multi-mode attributed graph $G = (V', E', \Lambda')$ such that $V' \subseteq V$, $E' \subseteq E$ and $\Lambda' \subseteq \Lambda$.*

Intuitively speaking, a good cluster should

- contain vertices having a high density of edges in as many modes as possible, and
- contain attributes such that the vertices V' have a large mutual similarity in as many attributes as possible.

Based on a multi-mode attributed subgraph, we can define a multi-mode attributed cluster of $\mathcal{G}$ as a special case of a subgraph of $\mathcal{G}$. Informally, a cluster is a subset of vertices having a subset of modes and having a subset of attributes, such that all edges of the original graph $\mathcal{G}$ are preserved. Formally:

Definition 18 (Multi-Mode Attributed Graph Cluster)**.** *Let $\mathcal{G} = (V, E, \Lambda)$ be a multi-mode attributed graph. Let $V_C \subseteq V$ be a subset of vertices, let $\mathcal{M}_C \subseteq \mathcal{M}$ be a subset of modes and let $\Lambda_C \subseteq \Lambda$ be a subset of attribute functions.*

A cluster $C = (V_C, \mathcal{M}_C, \Lambda_C)$ is a subgraph $(V_\mathcal{G}, E_\mathcal{G}, \Lambda_\mathcal{G}) \subseteq \mathcal{G}$ such that:

- $V_\mathcal{G} = V_C$,
- $\Lambda_\mathcal{G} = \Lambda_C$,
- $E_\mathcal{G} = \{(v_i, v_j, k) \in E | v_i, v_j \in V' \wedge k \in \mathcal{M}_C\}$.

Example 5. *Returning to our example in Figure 3.1, a possible cluster is defined by users Betty, Danie and Eric using attribute* location*, mode* friends *and mode* coauthors*. The resulting cluster*

$$C_1 = (\{\mathit{Betty}, \mathit{Danie}, \mathit{Eric}\}, \{\mathit{friends}, \mathit{coauthors}\}, \{\mathit{location}\})$$

is depicted in Figure 3.2a. Following Definition 18, cluster C_1 inherits all edges that exist in the corresponding subgraph of $\mathcal{G}$ (c.f. Figure 3.1). Clearly, Betty, Danie and Eric are not particularly well connected in the selected modes and attribute: In terms of spatial location, they are very distant from each other, they are entirely unconnected in the friendship network, and only share one co-authorship link. Intuitively, we would much rather like to find a qualitatively better cluster, such as shown in Figure 3.2b. This cluster, which contains users Alfred, Betty and Danie, and uses attribute salary*, mode* colleagues *and mode* coauthors *shows a much higher connectivity: the three selected users form a clique in both selected modes and they are very similar in terms of their salary attribute. We can argue that this cluster shows a group of users having a particular strong connection in their friendship and spatial proximity networks, while also having similar age.*

The challenge of this work is to find such high-quality clusters, i.e., vertices $V_C \in V$, modes $\mathcal{M}_C \subseteq \mathcal{M}$ and attributes $\Lambda_C \subseteq \Lambda$, such that all vertices in V_C are strongly connected in all modes M_C and highly similar in all attributes Λ_C. Such a set of clusters is denoted as a clustering.

Definition 19 (Multi-Mode Attributed Graph Clustering)**.** *Let $\mathcal{G} = (V, E, \Lambda)$ be a multi-mode attributed graph. A clustering $\mathcal{C} = \{C_1, ..., C_K\}$ is a set of clusters of $\mathcal{G}$.*

A challenge of finding a useful clustering $\mathcal{C}$ is to avoid redundancy. In addition to the high quality cluster $C_2 := (\{\text{Alfred, Betty, Danie}\}, \{\text{Colleagues}$
$, \text{Coauthors}\}, \{\text{Salary}\})$, we may also find $C_3 := (\{\text{Alfred, Danie}\}, \{\text{Colleagues, Coauthors}\}, \{\text{Salary}\})$ having a high quality. However, both clusters are highly redundant, sharing the same modes and attributes and only differing in one user. Thus, the information of one cluster is much lower given the other cluster. Our approach addresses this challenge of minimizing redundancy between overlapping clusters by considering the information of one cluster given all other clusters. This information theoretic approach is described in the next section. Besides, there are edges connecting these clusters that are not included in any other cluster of a

clustering $\mathcal{C}$. Further, many attributes of nodes from the full-dimensional subspace are not assigned to any cluster. We define these areas as the non-cluster area of an attributed graph in Definition 20, which consists of the elements lying outside the structure and attribute space of a cluster.

Definition 20 (Non-Cluster Area in an Attributed Graph). *Let $\mathcal{G} = (V, E, \Lambda)$ be a multi-mode attributed graph and let $\mathcal{C}$ be a clustering of $\mathcal{G}$. For each mode $m_n \in \mathcal{M}$, the set of non-clustered edges*

$$U_n := \{(v_i, v_j, n) \in E|\} \setminus \bigcup_{k=1, m_n \in M_{C_k}}^{K} \{(v_i \in V_{C_k}, v_i \in V_{C_k}, m)\}$$

is defined as the set of all edges of $\mathcal{G}$ in mode m_n that do not appear in any of the clusters C_k. Further, we define U_F as the non-cluster area in the attribute matrix F:

$$U_F = \{F_{ij} | 1 \leq i \leq |V|, 1 \leq j \leq T\} \setminus \bigcup_{k=1, \lambda_j \in \Lambda_k} \{F_{ij} | v_i \in V_{C_k}\}.$$

as all the attribute values which do not appear in any cluster. All of the information $U_n, 1 \leq n \leq M$ and U_F that is not captured by clustering $\mathcal{C}$ is denoted as $U(\mathcal{C})$.

Example 6. *Reconsider the example of Figure 3.1 and a clustering $\mathcal{C} = C_1, C_2$ consisting of only the two clusterings shown in Figure 2. For the first mode m_1 which corresponds to the* Friends *network, the set of unclustered edges U_1 contains every single edge of the friends network because only the first cluster C_1 uses mode m_1, but does not contain any edges in this mode. For the third mode m_3 corresponding to the* Coauthors *network, the set of unclustered edges U_3 consists only of the edge between* Chris *and* Eric*. This is the only edge in the* Coauthors *network that is not captured by cluster C_2. The set U_F contains all the attribute values that are not captured by clusters C_1 and C_2, i.e., the location of* Alfred *and* Chris*, the salary of* Chris *and* Eric *and the age of all users.*

In the next section, we present an information theoretic approach to compress the information contained in a clustering $\mathcal{C}$ and its corresponding non-clustered regions $U_i, 1 \leq i \leq m$ and U_F.

3.1.2 Coding Scheme

As one core part of our approach, this section proposes a coding scheme to minimally describe a clustering of a multi-mode attributed graph using the paradigm of Minimum Description Length (MDL) [170]. As a lossless compression, the MDL principle follows the assumption that the less coding length we adopt to describe the data, the more knowledge we can gain from it. To compress data, MDL describes potentially large fractions of data, by relatively small data models. To apply MDL to our problem of clustering multi-mode attributed graphs, we will employ clusters as data models to compress the data. For an intuition of this idea, consider the following example.

Example 7. *Assume a single mode graph having 1,000,000 edges. To represent such a graph, we can use an adjacency list, exatcly one million edges. This representation allows to describe any network, regardless of its topology. Now, how can we compress this graph by exploiting its topology? Assume the network contains a set of 500 vertices forming a clique. We can store the information about this clique, using the set of 500 vertices. Then we no longer need to explicitly store the* $500 \cdot 499 = 249,500$ *edges which are already described by the clique model. In this case, we have significantly reduces the storage cost from* $1,000,000$ *vertex pairs to* $750,500$ *vertex pairs and a set of* 500 *vertexes describing the clique model. What if we don't have a full clique? Assume another set of 500 vertices having an edge density of* 90%*, thus having* $224,550$ *actually out of* $249,500$ *possible edges. Using the concept of minimum description length [170], we know that the subset of these edges can be uniquely described using only a fraction* $e(0.9)$ *of the full information, where* $e(0.9) = -0.9 \cdot \log_2 0.9 - 0.1 \cdot \log_2 0.1 = 0.469$ *has to be stored. Thus, we can describe these* $224,550$ *edges at only* 46.9% *of their description cost, by incurring an additional model information about the dense cluster of the corresponding 500 vertices. Using the same principle, we can identify further cluster models, which yield a reduction in edge description that outweighs the additional information required to describe the cluster model.*

This example illustrates how we can use clusters to minimize the description length of a network. Our aim to is to use the MDL principle to find models which minimally describe the network. By allowing MDL to only use clusters as data models, we are guaranteed to obtain high quality clusters as a result.

Formally, the quality of a model can be identified from Equation 3.1 , where $L(M)$ denotes the coding length for describing model M and its parameters, while $L(D \mid M)$ represents the cost of coding data D under model M.

$$L(M, D) = L(D \mid M) + L(M) \tag{3.1}$$

In our case, the data D corresponds to the multi-mode attributed graph $\mathcal{G}$ and the describing model M corresponds to a clustering $\mathcal{C}$ of $\mathcal{G}$, yielding:

$$L(\mathcal{C}, \mathcal{G}) = L(\mathcal{G} \mid \mathcal{C}) + L(\mathcal{C}). \tag{3.2}$$

The length of the data under the model $L(\mathcal{G} \mid \mathcal{C})$ can be described as the total coding cost of all individual clusters and the non-cluster area in the multi-mode attributed graph $\mathcal{G}$. The length of the model $L(\mathcal{C})$ is the coding length of assignment of vertices, modes and attributes to each cluster, and the coding cost of parameters. In the following, we elaborate in detail on both the data description cost $L(\mathcal{G} \mid \mathcal{C})$ (Section 3.1.2) and the model description cost $L(\mathcal{C})$ (Section 3.1.2) that are necessary to compress an attributed multi-mode graph. Then, in Section 3.1.2 we provide an intuition why minimizing the data description cost of our coding scheme will yield a high-quality clustering.

Data Description Cost $L(\mathcal{G} \mid \mathcal{C})$

Suppose K clusters $\mathcal{C} = \{C_1, C_2, ..., C_K\}$ are discovered from an attributed graph $\mathcal{G}$. The data description cost $L(\mathcal{G} \mid \mathcal{C})$ requires the coding cost $CC(C_i)$ of all clusters C_i and the coding cost $CC(U(\mathcal{C}))$ of the non-clustered information:

$$L(\mathcal{G} \mid \mathcal{C}) = \sum_{i=1}^{k} CC(C_i) + CC(U(\mathcal{C})) \tag{3.3}$$

In the following, we will describe the data description cost.

Coding cost of a cluster $CC(C_i)$: Compressing an attributed graph requires compressing its adjacency matrices $A^n, 1 \leq n \leq M$ and its attribute matrix F. A cluster $C_i = (V_{C_i}, \mathcal{M}_{C_i}, \Lambda_{C_i}) \in \mathcal{C}$ is represented by the adjacency sub-matrices $A^m_{C_i}, m \in \mathcal{M}_{C_i}$ and the subset attribute matrix F_{C_i}. Consequently, the coding cost of the cluster C_i is the sum of the structural coding cost $CC(A^m_{C_i})$ of each adjacency matrix and the attribute coding cost $CC(F_{C_i})$, is:

$$CC(C_i)) = \sum_{m \in \mathcal{M}_{C_i}} CC(A^m_{C_i}) + CC(F_{C_i}) \tag{3.4}$$

For a good clustering, a cluster C_i should be composed of densely connected vertices which is equivalent to having a high probability of '1's in subset adjacency matrix $A^m_{C_i}$. The average coding cost of the entries in matrix $A^m_{C_i}$ is lower bounded by its entropy. Because we assume $\mathcal{G}$ to be an undirected graph, we only need to encode the entries of the upper triangular matrix. The coding cost of the structural information of the cluster C_i is described in Equation 3.5 , where $p^m_1(C_i)$ and $p^m_0(C_i)$ denote the probability of 1s and 0s in the adjacency matrix $A^m_{C_i}$ respectively. Further, $n^m_{C_i} = |V_{C_i}| \cdot (|V_{C_i}| - 1)/2$ refers to the number of entries in upper triangular matrix of $A^m_{C_i}$.

$$CC(A^m_{C_i}) = -n^m_{C_i} \cdot (p^m_1(C_i) \log_2 \cdot p^m_1(C_i) + p^m_0(C_i) \cdot \log_2 p^m_0(C_i)) \tag{3.5}$$

Secondly, we need to encode the attribute matrix F_{C_i}. For this purpose, we discretize each attribute domain $\theta_i, 1 \leq i \leq T$ into a finite set of discrete values $\theta'_i = \{\theta'_{i1}, ..., \theta'_{i|\theta'|}\}$.[1] After this discretization, the attributes of each node in C_i are represented by an element of the cross product of all discretized domains. Let $W := \bigtimes_{1 \leq i \leq T} \theta'_i$ denote the set of all possible attribute combinations, then

$$p(w \in W) = \frac{\sum_{v \in V_{C_i}} I(v, w)}{|V_{C_i}|}$$

denotes the fraction of nodes in C_i having attribute values corresponding to the discretized values in w. Here, $I(v, w)$ is an indicator function that returns one if the discretized attribute values of vertex v equal the values of w, and zero otherwise. Then, the coding cost of the attribute information of cluster C_i can be calculated as follows.

$$CC(F_{C_i}) = -|W| \cdot \sum_{w \in W} p(w) \cdot \log_2 p(w) \tag{3.6}$$

Being able to compute $CC(A^m_{C_i})$ and $CC(F_{C_i})$ using Equation 3.5 and Equation 3.6, respectively, we can substitute Equation 3.4 to obtain the coding cost for a single cluster C_i. Performing this computation for each $C_i \in \mathcal{C}$ yields the coding cost of

[1] For example, for numeric values, we can use a simple min-max or z-score normalization [134] and use a eight bit representation of the normalized values, for geo-spatial locations we can apply a quad-tree based decomposition.

all clusters, which is the first summand in obtaining $L(\mathcal{G} \mid \mathcal{C})$ following Equation 3.3. For the second summand, we next present a coding scheme to compute the coding cost of the non-clustered part $CC(U(\mathcal{C}))$ of clustering $\mathcal{C}$.

Coding cost of the non-cluster area $CC(U(\mathcal{C}))$: Following Definition 20, the non-cluster area of clustering $\mathcal{C}$ consists of the information in the non-clustered edges U_n for each mode $m_n \in \mathcal{M}$ and the non-clustered attribute values U_F. Consequently, the coding cost of the non-cluster area $CC(U(\mathcal{C}))$ is the sum of the structural coding costs $CC(U_n)$ for all $m_n \in \mathcal{M}$, and the attribute coding cost $CC(U_F)$:

$$CC(U(\mathcal{C})) = \sum_{m_n \in \mathcal{M}} CC(U_n) + CC(U_F) \tag{3.7}$$

For each mode, we consider the set of edges in U_n using an adjacency matrix. Again, due to assuming non-directed edges, we have $N = |V| \cdot (|V| - 1)/2$ possible edges, and we let $p_1(U_n) = \frac{|U_n|}{N}$ denote the relative fraction of non-zeros in the upper triangle of this adjacency matrix. Intuitively, $p_1(U_n)$ corresponds to the probability that a randomly picked edge in $V \times V$ is in U_n and $p_0(U_m) = 1 - p_1(U_n)$ corresponds to the counter probability. We obtain the following coding cost for each U_n.

$$CC(U_n) = -N \cdot (p_1(U_n) \cdot \log_2 p_1(U_n) + p_0(U_n) \cdot \log_2 p_0(U_n)). \tag{3.8}$$

Using the same attribute encode that was used for the cluster attribute matrices F_{C_i}, we again map each attribute domain θ_i to a discretized domain θ'_i. This yields a discretized version of the non-clustered attribute matrix U_F. For each attribute $\lambda_i \in \Lambda$, and each discrete domain value $v \in \theta'_i$, let $N(\theta_i = v)$ denote the occurrences of value v, and let $p^i_v = \frac{N(\theta_i = v)}{|V|}$ denote the probability of a random vertex having the discretized attribute value v in attribute λ_i. We encode each attribute value in U_F individually.

$$CC(U_F) = -\sum_{i=1}^{T} \sum_{v \in \theta'_i} |\theta'_i| \cdot p^i_v \cdot \log_2 p^i_v. \tag{3.9}$$

Now, we can solve Equation 3.4 by substituting each cluster coding cost $CC(C_i)$ by using Equation 3.5, and by substituting $CC(U_\mathcal{C})$ by using Equation 3.6, thus yielding the total data description cost $L(\mathcal{G} \mid \mathcal{C})$. Next, in order to solve Equation 3.3 to compute the total coding cost $L(\mathcal{C}, \mathcal{G})$, we need to compute the coding cost $L(\mathcal{C})$ for the clustering $\mathcal{C}$.

Model Cost $L(\mathcal{C})$

For encoding the model cost $L(M)$ of the attributed graph, each cluster $C = (V_C, \mathcal{M}_\mathcal{C}, \Lambda_C)$ will be compressed in three aspects: the assignments of each vertex, the assignments of attributes and the assignments of modes.

Coding cost of vertices assignment $CC_V(C_i)$: As overlapping is allowed in our proposed algorithm, a vertex can be assigned to multiple clusters. For each cluster, we adopt an assignment list to label the existence of vertices. When the vertex belongs to the cluster, the corresponding value in the list is set to 1, otherwise set to 0. Therefore, the coding cost of the vertices assignment for a cluster $CC_V(C_i)$ in a single mode m is lower bounded by its entropy as follows:

$$CC_V^m(C_i) = -|V| \cdot (p_1^m(C_i) \cdot \log_2 p_1^m(C_i) + p_0^m(C_i) \cdot \log_2 p_0^m(C_i)), \tag{3.10}$$

where $p_1^m(C_i)$ and $p_0^m(C_i)$ denote the probability of 1s and 0s in the assignment list of cluster C_i in mode m respectively, and $|V|$ is the number of vertices in the graph. Then the coding cost of the vertices assignments of the whole graph using all modes in $\mathcal{M}$ is the sum of cost for all clusters in all modes:

$$CC_V(\mathcal{C}) = \sum_{C_i \in \mathcal{C}} \sum_{m \in \mathcal{M}} CC_V^m(C_i)$$

Coding cost of attributes and mode assignments: In our proposed algorithm, the corresponding attribute and mode subspaces of clusters are also detected. Similarly, there is overlapping among the subspaces as well, which means an attribute or mode can be assigned to multiple clusters. Again, we use an assignment list of size T for the attribute-cluster matching, and an assignment list of size M for the mode-cluster matching. Again, we use the entropy to lower bound the coding cost of this matching. For the attribute assignment, we get

$$CC_F(C_i) = -T \cdot (p_1(C_i) \cdot \log_2 p_1(C_i) + p_0(C_i) \cdot \log_2 p_0(C_i)), \tag{3.11}$$

where $p_1 = \frac{|\Lambda_{C_i}|}{T}$ is the fraction of attributes used by C_i and

$$CC_\mathcal{M}(C_i) = -M \cdot (p_1(C_i) \cdot \log_2 p_1(C_i) + p_0(C_i) \cdot \log_2 p_0(C_i)),$$

where $p_1 = \frac{|\mathcal{M}_{C_i}|}{M}$ is the fraction of modes used by cluster C_i. Adding the three types of model coding costs we obtain the total model cost:

$$L(\mathcal{C}) = CC_V(\mathcal{C}) + CC_F(C_i) + CC_{\mathcal{M}}(C_i) \tag{3.12}$$

Finally, by using Equation 3.4 and Equation 3.12 for substitution in the right-hand-side of Equation 3.3, we obtain the total coding cost of a clustering $\mathcal{C}_i$.

Intuition and Discussion

First, one might argue that our coding scheme, by employing entropy as an objective function, will not only prefer very densely connected subgraphs, but will also prefer extremely sparsely connected subgraphs. For example, a cluster of vertices that have no connections in $\mathcal{G}$, will have an entropy of zero. However, adding such a loose cluster to our clustering will *not* decrease the overall entropy $L(\mathcal{G} \mid \mathcal{C})$ of the clustering, since all edges need to be encoded, possibly using the unclustered area $U(\mathcal{C})$. Ad absurdum, a clustering which only contains clusters having zero edges in all of their modes, will indeed have a cost of zero to encode all clusters. But in this case, the unclustered region $U(\mathcal{C})$ will still contain all edges, such that the coding cost of this region alone equals the coding length of the uncompressed graph $\mathcal{G}$, such that nothing is gained. Intuitively, a high quality cluster having a high density of edges, will allow a large number of edges to be encoded at a very low cost, thus decreasing the global cost $L(\mathcal{C}, \mathcal{G})$.

Now, the data encoding cost could be reduced to zero if each edge was described by an individual cluster, containing only the corresponding two connected nodes, and only having the corresponding mode of that edge. In that case, each cluster would be a two-node-clique, and the unclustered region would be empty, thus incurring a zero data description cost $L(\mathcal{G} \mid \mathcal{C})$. However, in this case where each edge is represented by its own cluster, the model cost would be extremely large, incurring a cluster encoding cost for every single edge, thus incurring at least as much cost as the uncompressed graph G. Summarizing, a clustering that minimizes the overall coding cost $L(\mathcal{C}, \mathcal{G})$ should consist of a small number of densely connected clusters.

3.1.3 Algorithm: IROC

In this section, we propose the algorithm IROC to heuristically efficiently detect overlapping clusters in attributed graphs. The initialization phase and the refinement phase are the two key steps in IROC. The initialization phase is again divided into two subroutines for a) creating initial graph substructures and b) finding their coherent attribute subspace. The refinement phase improves the quality of the initial clusters by a) removing redundant parts of the cluster and b) reassigning vertices between two clusters.

Initialization

Creating Initial Clustering: For each vertex $v_i \in V$ we create one initial cluster C_i by adding all vertices to C_i that are connected to v_i in any mode. Initially, each cluster uses all modes and all attributes, i.e.,

$$C_i = (v_i \cup \{v_j \in V | \exists k : (v_i, v_j, k) \in E\}, \mathcal{M}, \Lambda). \tag{3.13}$$

We iteratively and greedily choose K initial clusters from the set $\{C_1, C_{|V|}\}$ minimizing the description length of Section 3.1.2.
After obtaining a clustering $\mathcal{C}$ consistingExperiments on synthetic and real-world data show promising comparisons to state-of-the-art methods with respect to efficiency and effectiveness. of K initial clusters, we need to find the attribute and mode subspace of each of these clusters. Initially, we start having $\Lambda_C = \emptyset$ and $\mathcal{M}_C = \emptyset$. In this case, cluster C has a coding cost of zero, but an extremely large cost is attributed to the non-clustered edges and attributes following Equation 3.8 and Equation 3.9. Algorithm 1 summarizes the whole initialization step in pseudo-code. In Lines 1 to 6 the first initial K sets of nodes are selected. Then, for each set, the function *FindSubspace* (Algorithm 2) is called to assign modes and attributes to each set of nodes, thus yielding an initial clustering. We note that K is a parameter denoting the initial number of clusters. In Section 3.1.3 we propose heuristics to choose this parameter automatically, and our experiments show that both the run-time and quality of IROC are insensitive to the choice of K.

Algorithm 1. Initialization Phase

```
Input: Multi-Mode Attributed Graph $\mathcal{G} = (V, E, \Lambda)$
    Integer K
Output: Initial Clustering: $\mathcal{C} = \{C_1, C_2, ..., C_K\}$
 1: Construct $V$ subgraphs $S = \{s_1, s_2, ..., s_N\}$ as initial clusters;
 2: $\mathcal{C} \leftarrow \emptyset$;
 3: for 1 to $K$ do
 4:     $s_{best} \leftarrow$ Best Cluster $s \in S$ using Equation 1.
 5:     $\mathcal{C} \leftarrow \mathcal{C} \cup (s_{best}, \emptyset, \emptyset)$
 6: end for
 7: for $C = (V_C, \mathcal{M}_C, \Lambda_C) \in \mathcal{C}$ do
 8:     $C \leftarrow$ FindSubspace$(\mathcal{C}, V_C)$
 9: end for
10: return $\mathcal{C}$
```

Refinement

After the initialization step, our clustering $\mathcal{C}$ contains K clusters. These clusters were generated completely independent of each other. Thus, these clusters may contain *redundant* information. To remove redundancy, our refinement iterates two steps: the *merge step* and the *split* step. Briefly, the *merge step* attempts to merge two existing clusters into one, while the *split* step attempts to remove a vertex from a cluster to form a new cluster.

The refinement step is shown in Algorithm 3 in greater detail. First, the *merge step* finds a pair of clusters and merges these two clusters into a single cluster such that it yields the highest reduction of coding cost. These vertices are selected by

$$(C_1, C_2) = \underset{(C_1, C_2 \in \mathcal{C})}{\operatorname{argmin}} L(\mathcal{C} \setminus C_1 \setminus C_2 \cup \text{FindSubspace}(V_{C_1} \cup V_{C_2}), \mathcal{G}). \tag{3.14}$$

Here, function *FindSubspace* is invoked in Line 5 to find the attributes and modes of the new cluster, which might be different from the old ones. This merge is performed in Line 6, where the new cluster C_{merge} is added to $\mathcal{C}$ with the old clusters being removed.

In the *split step*, we try to split a single vertex from C_{merge} into a new (singular) cluster C_{split}. This single vertex is selected by

$$v_{best} = \underset{v \in C_{\text{split}}}{\operatorname{argmin}} L(\mathcal{C} \setminus C_{\text{merge}} \cup$$

Algorithm 2. FindSubspace($\mathcal{C}, V_C$)

```
Input: A clustering $\mathcal{C}$ and a set of nodes $V_C$
Output: A cluster $C' = (V_C, \mathcal{M}'_C, \Lambda'_C)$
 1: $\Lambda_C \leftarrow \emptyset$;
 2: $\mathcal{M}_C \leftarrow \emptyset$
 3: $CC = L(\mathcal{C}, \mathcal{G})$
 4: $CC_{old} \leftarrow \infty$
 5: $\mathcal{C}_{old} \leftarrow \mathcal{C}$
 6: while $CC < CC_{old}$ do
 7:     $\mathcal{C} \leftarrow \mathcal{C}_{old}$
 8:     $CC_{old} = CC$
 9:     $m_{best} \leftarrow$ Best mode $m \in \mathcal{M} \setminus \mathcal{M}_C$
10:     $C_{\mathcal{M}} \leftarrow (V_C, \mathcal{M}_C \cup m_{best}, \Lambda)$
11:     $\lambda_{best} \leftarrow$ Best attribute $\lambda \in \Lambda \setminus \Lambda_C$
12:     $C_\Lambda = \leftarrow (V_C, \mathcal{M}_C, \Lambda \cup \lambda_{best})$
13:     if $CC(\mathcal{C} \setminus C \cup C_{\mathcal{M}}) \geq CC(\mathcal{C} \setminus C \cup C_\Lambda)$ then
14:         $\mathcal{C}_{old} = \mathcal{C} \setminus C \cup C_{\mathcal{M}})$
15:     else
16:         $\mathcal{C}_{old} = \mathcal{C} \setminus C \cup C_\Lambda)$
17:     end if
18: end while
19: return $(V_C, \mathcal{M}_C, \Lambda_C)$
```

$$\text{FindSubspace}(C_{\text{merge}} \setminus v) \cup \text{FindSubspace}(\{v\}), \mathcal{G}). \tag{3.15}$$

This best vertex is selected in Line 7 of Algorithm 3. If any split reduces the overall coding length, then this split is performed in Line 9. If after any iteration invoking both *merge* and *split* steps, no overall reduction of coding length is achieved, then the algorithm terminates, returning $\mathcal{C}_{old}$.

Overall procedure

The overall procedure of IROC is shown in Algorithm 4. First, we automatically select K_s rough clusters and search their attribute subspace as described in the initialization phase. Then we calculate the similarity of every pair of clusters, and merge the two cluster with a minimum similarity. After that a new cluster C_{new} is formed and we find the subspace of it. Then we try to assign vertices from C_{new} to C_{split} under the control of MDL. And we consider C_{split} as a new cluster and recalculate the similarity of every pair of clusters. The merging process

Algorithm 3. Refinement

```
Input: Clustering $\mathcal{C}$
Output: Clustering $\mathcal{C}$
 1: $L_{old} = \infty$, $L = L(\mathcal{C}, \mathcal{G})$
 2: while $L < L_{old} \wedge iter < maxIter$ do
 3:     $\mathcal{C}_{old} = \mathcal{C}$, $L_{old} \leftarrow L$
 4:     Find pair of clusters $C_1, C_2 \in \mathcal{C}$
 5:     mergedCluster = FindSubspace($V_{C_1} \cup V_{C_2}$)
 6:     $\mathcal{C} \leftarrow \mathcal{C} \setminus C_1 \setminus C_2 \cup$ mergedCluster
 7:     $v_{best} \leftarrow$ best split vertex
 8:     if $L(\mathcal{C}, \mathcal{G}) > L(\mathcal{C} \setminus C\text{merge} \cup \text{FindSubspace}(C\text{merge} \setminus v) \cup$
        FindSubspace($\{v\}$), $\mathcal{G}$) then
 9:         $\mathcal{C} \leftarrow \mathcal{C} \setminus C\text{merge} \cup \text{FindSubspace}(C\text{merge} \setminus v) \cup \text{FindSubspace}(\{v\})$
10:         $iter++$
11:     end if
12: end while
13: return $\mathcal{C}_{old}$
```

continues iteratively and is ended when the coding cost of all clusters achieve its local minimum. Finally, K clusters with coherent attribute subspaces without redundancy are output.

Note as our refinement algorithm is not guaranteed to converge, we use a parameter maxIter to enforce termination.

Complexity Analysis

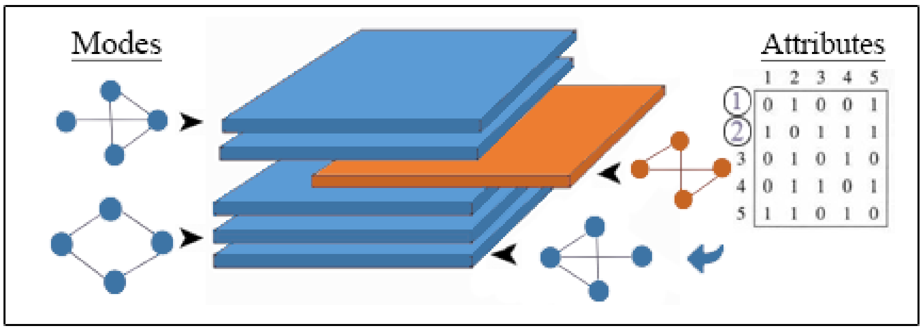
Assume a graph $\mathcal{G}$ with $|V|$ vertices, $|E|$ edges, T attributes and M modes. We first analyze the complexity of computing the coding length $L(\mathcal{C}, \mathcal{G})$. For each cluster C we need to count the edges within C in each mode $m \in \mathcal{M}$ to obtain the probabilities $p_1^m(C)$ required by Equation 3.5. Due to potential overlap of clusters, the total number of edges in all clusters is not bounded by the number of edges $|E|$ in all modes. In the worst case, where one mode forms a full clique of all vertices, the initial clustering of the initialization step will contain all vertices in each cluster, yielding a worst case of $|V| \cdot |E|$ edges in all clusters. However, assuming the all modes have sparse connectivity, this number decreases to $O(|E|)$. In addition, the coding of the attribute matrix (which is not assumed to be sparse) additionally requires $O(|V| \cdot T)$ time, yielding a $f_{code} := O(|E| + |V| \cdot T)$ time complexity to compute the coding length of $\mathcal{C}$.

Algorithm 4. IROC

```
Input: Attributed Graph $G$
Output: $K$ Clusters with Subspace $C = C_1, C_2, ..., C_K$
 1: Creating Subgraphs $C = C_1, C_2, ..., C_{K_s}$;
 2: for $i$ from 1 to $K_s$ do
 3:     Finding Subspace of $C_i$
 4: end for
 5: while Not Converge do
 6:     Calculate similarity of every pair of clusters;
 7:     Merge two most similar clusters as $C_{new}$;
 8:     Finding Subspace of $C_{new}$;
 9:     Assigning Vertices of $C_{new}$;
10: end while
11: return $C = C_1, C_2, ..., C_K$.
```

For the initialization phase of IROC, we greedily choose K clusters from the initial $|V|$ sets of clusters. For each such choice, we need to compute the coding length of $\mathcal{C}$, yielding a total cost of $O(K \cdot |V| + |K| \cdot f_{code}) = O(K \cdot (|E| + |V| \cdot T))$ complexity. Then, finding the mode and attribute subspace of each cluster requires, in the worst-case where all modes and attributes are selected, $O(T^2 + M^2)$ invocations of the coding length computation $L(\mathcal{C}, \mathcal{G})$. Overall, the initialization cost is in $O(K \cdot (|E| + |V| \cdot T + T^2 + M^2))$.

In the refinement step, we need to iterate $|\mathcal{C}|$ pairs of clusters to find the best pair to merge, invoking $L(\mathcal{C}, \mathcal{G})$ in each iteration, yielding a cost of $O(\mathcal{C}^2 \cdot |E| + |V| \cdot T)$. In the split step, we need to iterate, for the merged cluster C, over all $|V_C|$ vertices in C and perform a call of $L(\mathcal{C}, \mathcal{G})$. In the worst-case, the size of a cluster is in $O(|V|)$, yielding a worst-case run-time of $O(V \cdot (|E| + |V| \cdot T))$. This worst-case quadratic run-time seems prohibitive. But our experiments show, that in practice, this run-time scales nearly linear in $|V|$: the reason is that on real data, clusters (or communities) do not span $O(|V|)$ users. As connections are sparse, communities are relatively small compared to the full network size. Thus, our *IROC* algorithm is able to adapt to these small clusters, thus yielding a much cheaper cost of merging small clusters.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Figure 3.3: Tensor Construction: Five modes and one attribute.

Discussion

A parameter used by the *IROC* algorithm is the seed-size K, which significantly impacts the run-time of the algorithm. For a value of K choosen too large, e.g., for $K = |V|$, the quadratic runtime in the initial number of clusters will yield a run-time quadratic in $|V|$. For a value of K chosen too small, e.g., for $K = 1$, each iteration of the refinement step will run extremely fast, but the number of refinement steps required to obtain a high quality clustering (i.e., a low coding cost), will be extremely large. Per default, we recommend to set $K = \log_2 |V| \cdot \log_2 M \cdot \log_2 T$, which yields a reasonable seed of clusters in all our experiments. Our experiments leading to this recommendation are shown in Section 3.1.5.

3.1.4 IROC meets Tensor Factorization

As our experimental evaluation in Section 3.1.5 shows, a main drawback of the *IROC* algorithm proposed in Section 3.1.3 is the large number of iterations of the *refinement*. This drawback is the result of a large number of small initial clusters, which are iteratively merged and split. To address this problem, we next propose an alternative solution to obtain an initial clustering for the refinement step of our proposed *IROC* algorithm. Therefore, we exploit recent research on efficient factorization approximation of very large tensors [158, 105] which have become popular for clustering large network data. We show how to apply these algorithms to multi-mode attribute graph data, and then feed the resulting clusterings into our *IROC* algorithm, to further improve this clustering.

Attribute Mode Tensor: Following our definition of a M-mode attributed graph in Definition 16, an edge is defined by an element of $|V| \times |V| \times M$, which corresponds to a three-mode tensor. Thus, implicitly, the modes are already represented by a tensor. To incorporate information about an attribute function λ in this tensor, we generate a $|V| \times |V|$ similarity matrix for each attribute, which describes all pairs of nodes that are sufficiently similar to each other. More formally, we represent a multi-mode attribute graph $\mathcal{G} = (V, E, \Lambda)$ by the following three-mode $|V| \times |V| \times (M + T)$ tensor $\mathcal{T}^{\mathcal{G}}$:

$$\mathcal{T}^{\mathcal{G}}{}_{i,j,k} = \begin{cases} 1 & \text{if } k \leq M \wedge (i,j,k) \in E \\ 0 & \text{if } k \leq M \wedge (i,j,j) \notin E \\ 1 & \text{if } k > M \wedge d_{k-M}(\lambda_{k-M}(V_i), \lambda_{k-M}(V_j)) \leq \epsilon_{k-M} \\ 0 & \textit{otherwise} \end{cases} \tag{3.16}$$

Here, $d_i : \theta_i \times \theta_i \mapsto \mathbb{R}$ is a distance function defined on domain θ_i, and ϵ_i is a distance threshold for d_i such that any pair of vertices v_1 and v_2 is considered similar in attribute i if $d_i(v_1, v_2) \leq \epsilon$. We note that this *sparse* similarity matrix $\mathcal{T}^{\mathcal{G}}{}_{\cdot,\cdot,M+i}$ of attribute function λ_i can, for most domains θ_i be derived efficiently using a similarity self-join on all attribute values of vertices V. Exemplary, if attribute i corresponds to a geo-location, i.e., $\theta_i \in \mathbb{R}^2$, we can perform a distance self-join [160] exploiting possible spatial index structures. Summarizing, Figure 4.4 illustrates our process of generating the tensor $\mathcal{T}^{\mathcal{G}}{}_{i,j,k}$ from mode-specific adjacency matrices and the attribute matrix F.

Tensor Factorization: Given the tensor $\mathcal{T}^{\mathcal{G}}$, we employ the ParCube [158] algorithm to obtain an initial clustering ParCube($\mathcal{T}^{\mathcal{G}}$. Each cluster ParCube$_i$ = (V_i, X_i) yields a set of vertices V_i and a set of third-mode values X_i, which we translate back to a mode $m \in \mathcal{M}$ and attribute functions $\lambda\Lambda$. Formally, each tensor cluster ParCube$_i$ = (V_i, X_i) induces the cluster

$$C_i = (V_i, \{m_k | k \in X_i\}, \{\lambda_k | k - M \in X_i\}).$$

These clusters form the initial cluster $\mathcal{C}$ that we can feed directly to the refinement phase of the *IROC* algorithm, thus replacing the initial matrix passed to Algorithm 3. Our adapted *IROC* algorithm using the ParCube algorithm in the initialization step will be referred to as *TF-IROC*. Intuitively, we expect the *TF-IROC* algorithm to run significantly faster than the *IROC* algorithm, because the initialization step is able to use efficient tensor factorization to obtain a "good" initial clustering, better than naive initial clustering employed by IROC. Also, the initial number of clusters is much smaller thus the refinement requires less iterations. At the same time, this initial clustering takes away freedom from the *IROC* algorithm by already being given a full cluster structure. In our experiments, that this loss of algorithmic freedom yields a significant loss of clustering quality, but allows the *TF-IROC* algorithm to be applied to very large datasets.

Table 3.1: Evaluation Overlapping Clusters of Synthetic Data Sets

Algorithms	#Clust = 2			#Clust = 3			#Clust = 5		
	NMI	Ω-Index	F1-Measure	NMI	Ω-Index	F1-Measure	NMI	Ω-Index	F1-Measure
IROC	**1**	**1**	**1**	**1**	**0.973**	**0.986**	**1**	**0.963**	**0.981**
TF-IROC	1	0.963	0.982	0.914	0.951	0.973	0.966	0.921	0.952
PICS	0.615	0.732	0.846	0.563	0.670	0.612	0.270	0.322	0.487
DB-CSC	0.595	0.826	0.859	–	–	–	–	–	–
BAGC	0.588	0.947	0.953	0.955	0.607	0.742	0.490	0.722	0.584
PARCUBE	0.764	0.732	0.628	0.422	0.532	0.615	0.412	0.466	0.481

3.1.5 Experimental Evaluation

In this section, we evaluate our proposed algorithms IROC and TF-IROC on both synthetic and real data sets and compare them to state-of-the-art competitors. All experiments were performed on a Intel Xeon E7-4870 v2 (60x2.3 GHz) server with 1 TB main memory, running Ubuntu Linux 15.10 with the 4.2 kernel. Our competitor approaches evaluated in this section are the following:

DB-CSC [81] is a density based clustering algorithm at detecting dense clusters with a coherent subspace of attributes. DB-CSC allows finding clusters that overlap in attributes and vertices.

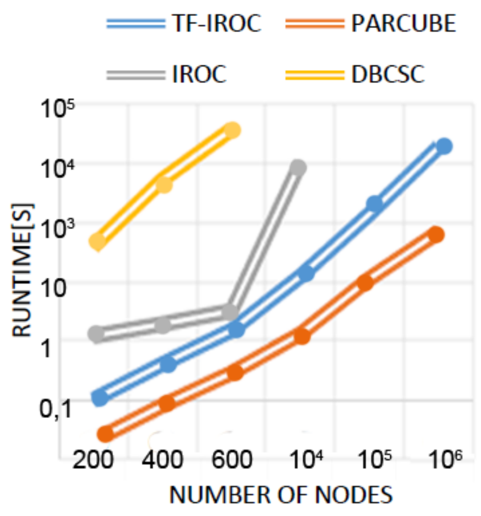
BAGC [196] is a probability model based attributed graph partition method, which requires the resulting clusters to be non-overlapping.

PICS [12] is a compression-based, parameter-free algorithm that clusters both vertices and binary attributes. This algorithm also returns non-overlapping clusters.

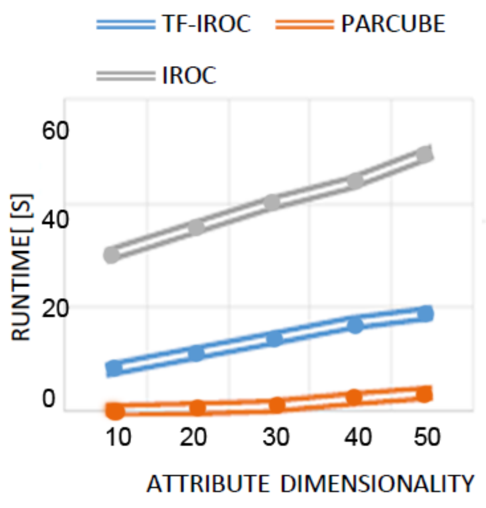
ParCube [158] is an tensor factorization algorithm which we also employ in the the initialization step of the TF-IROC algorithm. The ParCube algorithm is an approximate solution for the parafac tensor decomposition [89]. In contrast to TF-IROC, the pure ParCube algorithm does not use the refinement step of TF-IROC, thus terminating directly after the tensor factorization step. This algorithm requires to specify a sampling factor s which we set to $s = 1000$ in all experiments, as after this value we found no more increase in clustering quality. ParCube also requires to specify a threshold τ to specify the point at which no more latent factors are added to the model. Thus, the lower τ, the more latent factors will be used by ParCube.

To evaluate these solutions in terms of accuracy, we use the following evaluation criteria.

F1-Measure. Due to the multiple clustering assignments, we adopt F1-Measure to evaluate these algorithms on clustering vertices. F1-Measure is computed as the harmonic mean of Precision and Recall. Precision measures the accuracy of the detected clusters and Recall measures whether all clusters are detected.
NMI. This criteria is basing in information theory and was extended by Lancichinetti et al. [124] to account for overlap between communities.
Omega Index is the overlapping version of the Adjusted Rand Index (ARI) [207]. It is based on pairs of nodes in agreement in two covers. Here, a pair of nodes is considered to be in agreement if they are clustered in exactly the same number of communities.



(a) Varying Network Size



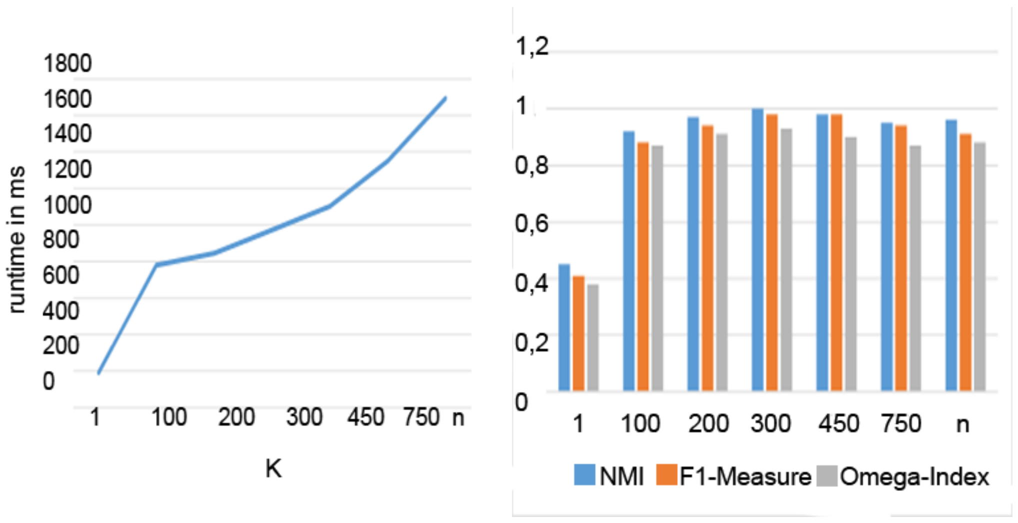
(b) Varying the number of modes and attribute

Figure 3.4: Synthetic Runtime Experiments

Evaluation on Synthetic Data

Synthetic Data Generation: For our scalability experiments, we use synthetic datasets, where we control the overlap of clusters in each mode. The parameters and default values of this synthetic dataset are described as follows. The parameter $\#\text{Clust} = 5$ denotes the number of artificial clusters in the dataset. Per default, $n = \#\text{Clust} \cdot 1000$ vertices are generated, each vertex has a probability of $p = 0.5$ to belong to any of the clusters. This allows a vertex to belong to any number of clusters. For instance, for $\#\text{Clust} = 2$, we have two clusters, $n = 2000$ vertices, and approximately 500 of the vertices belong to both clusters, 500 belong to either one cluster, and 500 belong to none of the two clusters. In each of the $m = 3$ modes, the same assignments from nodes to clusters is performed independently. For each mode k, an edge (v_i, v_j, k) belongs to $\mathcal{G}$ with a probability of $d_b = 0.1$, if nodes (v_i) and v_j do not appear together in any artificial cluster in mode k, or with a probability of $1 - d_c^{\#\text{Clust}(v_i,v_j,k)}$, where per default $d_c = 0.2$ and $\#\text{Clust}(v_i, v_j, k)$ is the number of clusters of mode k containing both v_i and v_j. As an example, for $\#\text{Clust} = 2$, an edge between a pair of vertices contained in exactly one cluster, has a probability of $1 - 0.2^1 = 0.8$ to exist, whereas an edge between a pair of vertices contained in both clusters would have a probability of $1 - 0.2^2 = 0.96$ to exist. Furthermore, each vertex is assigned $T = 3$ attributes. All attributes functions λ_i map a vertex to a binary domain $\theta_i = [0, 1]$. Attributes values are generated randomly, such that a pair of vertices that appears in a large number of clusters in a large number of nodes has a higher probability of having identical attribute values. If a pair of vertices is not contained in any cluster in any mode, then these vertices have a probability of 0.5 of having the same attribute value in each attribute. If the pair of vertices appears in a total of $\#\text{Clust}(v_i, v_j) := \sum_{i=1}^{m} \#\text{Clust}(v_i, v_j, i)$ clusters over all modes, then the probability of sharing the same attribute values is $1 - 0.5 * a_c^{\#\text{Clust}(v_i,v_j)}$ having $a_c = 0.8$. For example, two vertices sharing exactly four clusters of all modes would have, for each attribute, a probability of $1 - 0.5 * 0.8^4 = 0.795$ to share the same attribute. Attribute values are chosen iteratively, starting with random attribute values at an initial vertex.

Our synthetic datasets allow us to scale the number of ground-truth clusters, the number of modes and attributes, as well as the size of the network in a sandbox. Furthermore, by construction of the synthetic data-set, we have a clean definition of a ground-truth clustering, allowing us to measure the effectiveness of different approaches in terms of finding these clusterings.



(a) Runtime Impact (b) Accuracy Impact

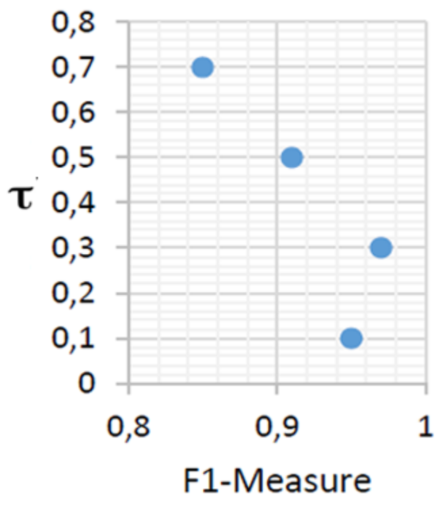
Figure 3.5: Effect of the initial number of clusters K.

Clustering Accuracy: Table 3.1 presents efficiency results, in terms of finding the artificial ground-truth clusters. The dataset having only #Clust = 2 clusters in each mode, IROC manages to perfectly separate the two ground-truth clusters in each mode. Our more efficient algorithm TF-IROC is reasonably close to this result. The algorithm PICS outputs a much larger number of clusters in each mode, which split into two big clusters and several small ones. BAGC finds two clusters, but is unable to detect any overlap. DB-CSC outputs an extremely large number of clusters in each mode where clusters contain less than ten vertices. We run DB-CSC with $\epsilon = 0.5$, $k_{min} = 4$, $min_{pts} = 5$,$r_{obj} = 0.1$,$r_{dim} = 0.1$ and $s_{min} = 1$, which correspond to the best parameter setting we could find after some parameter tuning. Finally, the ParCube approach which uses only the tensor factorization approach without the greedy optimization beformed by IROC, returns For the synthetic data set having #Clust = 3 clusters, there exists overlap between all three clusters. IROC and TF-IROC still achieve extremely high accuracy, allowing to nearly perfect reconstruct the three overlapping clusters of each mode. PICS outputs an average of six clusters per mode, without overlapping. BAGC produces three clusters, but again cannot find any overlap. DB-CSC did not terminate even after adjusting its six parameters several times and running the algorithm several days. The last synthetic data set, having #Clust = 5 clusters, is more complex than the other two synthetic data sets with five mutually overlapping clusters. Again, IROC performs the best, always finding five clusters and their respective overlap, only having a few individual vertex-cluster mismatches. TF-IROC follows closely achieving a slightly less accurate clustering. The clustering quality of PICS and BAGC deteriorate, since neither of these algorithms can find overlapping clusters. Thus, any vertex contained in more than one cluster must be assigned to a single cluster, thus creating a large disparity. From these experiments we conclude, that non-overlapping clustering solutions such as PICS and BAGC are not applicable in finding useful results in a setting where clusters overlap. Since these algorithms essentially solve a much easier problem setting, we will exclude PICS and BAGC from further evaluation.

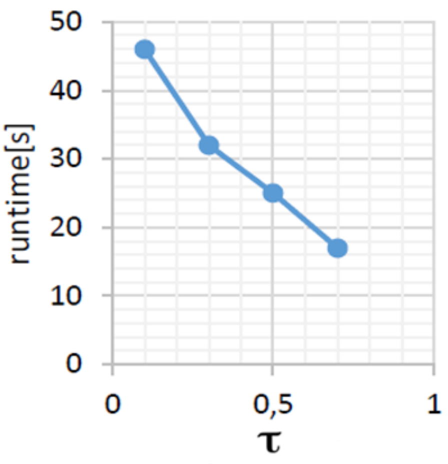
Efficiency: Next, we evaluate the runtime of IROC, TF-IROC and DB-CSC. As explained before, we no longer evaluate the algorithms PICS and BAGC, as these algorithms are unable to find overlapping clusters, thus yielding extremely low quality results in our settings. We vary both the number of vertices and the number of modes, which are shown in Figure 3.4.

In Figure 3.4a, we use a synthetic dataset having #Clust = 2 clusters. We scale the number of vertices per cluster from 200 to $1M$. The DB-CSC algorithm was only scaled to 1000 nodes due to its excessive run-time. Our IROC algorithm can be scaled to ten-thousands vertices, but the initialization step and the large number of initial clusters prohibit any further scaling. Our TF-IROC fills this gap, allowing to scale the this dataset to millions of vertices in reasonable time. We also see that PARCUBE has the fastest run-time. However, we see in Table 1, that due to the lack of clustering optimization performed by IROC, this pure tensor factorization approach yields a much lower clustering.

In Figure 3.4b, we simultaneously increase the number of modes M and attributes T from 10 to 50, keeping the number of vertices at 2000. While the pure tensor factorization outscales TF-IROC by a factor of 10; all algorithms scale linear in the number of attributes and modes dimensions.



(a) Accuracy Impact



(b) Runtime Impact

Figure 3.6: Effect of cutoff threshold τ

Initialization Parameters: The initialization steps of IROC and TF-IROC require to specify the number of initial clusters. IROC (c.f. Section 3.1.3) specifies this parameter directly. For TF-IROC, the number of clusters corresponds to the number of latent factors, which is controlled by the threshold τ. In this section, we show that these parameters do not require any fine-tuning or deep knowledge of the underlying dataset. Figure 3.5 evaluates this parameter K of IROC. We see in Figure 3.5a that for $K \geq 100$, the run-time scales linear in K. For $K = 1$, we see an extremely low run-time, due to a too-low number of clusters which are unable to capture the topology of the data. This yields a low clustering quality as shown in Figure 3.5b. Having $k \geq 100$ of initial clusters, the quality no longer improves. For all our experiments, choosing k logarithmic in the number of vertices $|V|$, the number of modes M and the number of attributes T yielding good results. Figure 3.6 shows the threshold parameter τ of the ParCube algorithm used to control the number of latent factors. We can see in Figure 3.6a that for any value of $\tau \in [0.1, 0.7]]$ the result quality is fairly high, having a maximum at around $\tau = 0.3$ in our default setting. For $\tau > 0.3$, the loss of quality is attributed to a small number of latent factors, and thus, again, a too-small number of initial clusters. For $\tau < 0.3$, we obtain a large number of clusters. But since ParCube optimizes a different objective function than MDL, these clusters are over-fitted to non-important latent factors, creating a high distance to the ground-truth clustering

		#Attributes	#Nodes	#Edges
Facebook	"686"	62	170	3312
Twitter	total	0	81306	1768149
	"356963"	247	65	1232
Google+	"1004"	150	891	4178
Youtube	total	0	1134890	2987624
Amazon	total	0	403394	3387388
Gnutella	total	0	62586	147892

Table 3.2: Sizes of all used real data sets.

which IROC can not repair without running into local minima. Again, we propose to choose τ in a way that the number of resulting latent factors (and thus clusters) follows the same rule-of-thumb described for parameter K. Figure 3.6b shows that the run-time is linear in the number of latent factors corresponding to the choice of τ. Finally, we note that IROC and TF-IROC may produce a number of clusters different to the initial number of clusters. Thus, the parameters K and τ are not a classic user-specified parameter requiring a-priory knowledge of the number of clusters in the data. The algorithm will try to find the real number of clusters iteratively, without any domain knowledge of the data.

In the two Fig. 3.6a and Fig. 3.6b we show how the implicit given threshold affects (a) clustering quality (b) the runtime of the TF-IROC. The threshold is used for pruning the result of the tensor factorization, giving meaning to the latent factors of the PARCUBE result. For a very small threshold, it is interesting to observe that the cluster quality decreases. This is because the tensor factorization follows a different optimization target than MDL. But we note again that the tensor factorization is only used to obtain an initial clustering, from where our IROC approach can iterate to improve the clustering. Consequently, the initial clustering does not need to be perfect and thus, we propose to use a fairly high threshold.

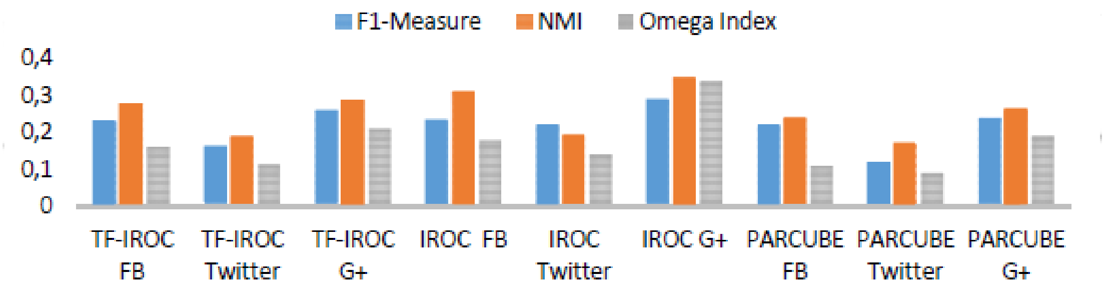


Figure 3.7: Evaluation criterias on Facebook, Google+ and Twitter Ego-Networks.

Real Data Sets

We use six SNAP [136] data sets as described in Table 3.2. The first three of them, Facebook, Twitter and Google+ are ego-networks having overlapping ground-truth clusters, called friendship circles. Since we are not able to match the identities of users between different networks, we only use a single mode in each of the real data sets. However, solutions for the problem of linking identities between users of different social networks has recently become a vivid research topic [130, 142]. To compensate for the lack of network modes, each of these networks has a large number of anonymized attributes. The other three real data sets are mainly for scaling experiments and only provide unlabeled network data.

Runtime Evaluation: The runtime on large real networks was evaluated using the unlabeled combined ego-networks of Facebook and Twitter and adding three other SNAP data sets: com-Youtube, amazon606 and Gnutella. For our approaches, only TF-IROC was evaluated as IROC was not able to scale to these large networks. We compare TF-IROC to ParCube, which is also used by TF-IROC in the initialization phase. We ran TF-IROC and PARCUBE in their parallelized version on all 60 cores. For each dataset, we set the sampling size s of ParCube to 20% of the original data. Figure 3.8 shows the results for these real-data sets. On all datasets, TF-IROC exceeds the run-time of ParCube by orders of magnitude. In particular, the iterative re-adjustment of clusters of the refinement step of TF-IROC is the bottleneck. Still, we see that TF-IROC still runs in acceptable time for our utilized social networks, while allowing to find more useful and representative clustering results, as we will see in the next experiment.

Quality Evaluation: The quality of the clustering of IROC, TF-IROC and its used tensor factorization ParCube [158] is given in Figure 3.7. For this experiment, we only used the data three datasets for which we have friendship circles that we use as a ground-truth for this dataset. For Facebook we used the ego-network"686", for Twitter the Ego-network "356963" and for Googleplus the one starting with "1004". The gold label necessary for all the measurements F1-measure, NMI and Ω-Index was extracted from given Circles of Friends included in the data and provided by the authors of [136]. These circles contain overlapping nodes, but not all nodes are labelled, thus explaining the overall low clustering quality. Figure 3.7 shows that the computationally expensive basic IROC yields the best clustering results. TF-IROC yields significantly worse results, but trades this loss in quality for a run-time applicable to large graph data. The ParCube algorithm yields the worst clustering quality.

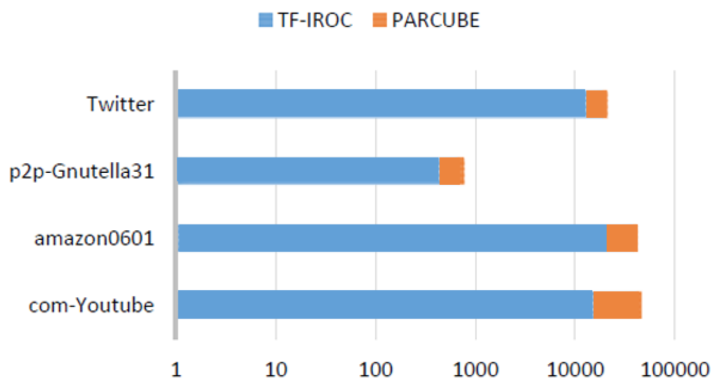


Figure 3.8: Runtime for large unlabeled real data sets

3.1.6 Related Work and Discussion

The problem of clustering vector data has been studied widely [195]. Our proposed algorithms IROC and TF-IROC are designed for clustering multi-modal attributed graphs by allowing not only overlapping in the network structure but also in the attribute subspace. The related work therefore comprises two parts: (multi-mode) attributed graph clustering and overlapping community detection methods in networks.

Attributed Graph Clustering

Guennemann et al. propose Gamer [82] and DB-CSC[81], which combine dense subgraph mining like DBSCAN [62] with subspace clustering. Both Gamer and DB-CSC allow overlapping in the attribute space and consider redundancy as all density-based approaches but are both very slow, use many parameters and the attributes supported must be numerical. PICS[12] is a parameter-free algorithm based on the MDL principle. It is able to mine cohesive clusters from an attributed graph with similar connectivity patterns and homogeneous attributes. However, it can not detect any overlapping in the attribute space and it clusters the vertices and attributes separately. Partition-based methods are, for example, BAGC [196] which base on an bayesian probabilistic model to partition attributed graphs. Another partitioning approach is from Zhou et al. [210]. It augments graphs by considering each attribute as a vertex. A random walk is utilized on the augmented graph to create a unified similarity measure to combines structural and attribute information. Obviously, these partition-based methods can not detect any overlapping of network clusters nor do they find any coherent attributed subspaces. Considering multi-mode or heterogeneous graphs, Sun et al. [179] propose a model-based method to clustering heterogeneous information networks, which contain incomplete attributes and multiple link relations. Also marginally related to our method are the approaches [177][141] and [185] achieving numerous small cohesive subgraphs, which aim to discover a correlation between node attributes and small subgraphs.

Detecting Overlapping Communities

The problem of finding overlapping network structures has been widely neglected, despite its applications in overlapping community detection in social networks. The fastest approaches exploit efficient matrix- and tensor factorization, for which efficient solutions have been recently proposed. [158][105]. Zhang et al. [208] propose a soft clustering algorithm based on matrix factorization. The assignment of each vertices is stored as probability in a matrix with a number of dimensions equal to the number of the community. The authors achieve the overlap of the communities by fuzzy allocation. Another approach using matrix factorization is [199] that works for a very large amount of data and makes the oberservation that overlapping network structures are actually denser than the others. To the best of our knowledge tensor factorization was not explicitly used in this approach as a clustering strategy but it was mentioned that it corresponds to co-clustering, which is always overlapping [157]. A key problem of finding overlapping clusters is how to assign a vertex to multiple labels. In a first instance, [73] reveals overlapping phenomena of complex networks in nature, and achieves overlapping communities by seeking k-cliques containing overlapping vertices. CONGA[77], proposed by Gregory, is an algorithm which aims to detect overlapping communities by iteratively calculating two betweenness centrality based concepts.In order to speed up the algorithm CONGA, the author proposes an algorithm named CONGO[78] by calculating local betweenness instead of global betweenness. Both algorithms need to at least set the number of clusters as input parameters. However, none of these algorithms are able to cluster networks having more than one mode, and having attributes associated with vertices. Finally, Non-overlapping community detection are widespread and similarities can be found in [156][192]. For a survey on overlapping community detection consider [194].

3.2 CXprime - Automated Clustering Using Structure Primitives

Real world data from various application domains, such as social networks, bioinformatics and neuronal networks can be modeled as graphs. Specific topological structures like triangles and stars represent meaningful characteristic relationships among subsets of nodes. Specifically, in [173] the authors introduced a transitivity attribute which is calculated from the fraction of triangles in all node triplets. As an indispensable condition for small-world networks, high transitivity implies more triangles in a graph. Moreover, the authors in [109] characterize graphs with a power-law degree distribution - a significant feature of scale-free networks - by a very low degree of most vertices combined with a high degree of only very few vertices. Therefore, hubness plays a pivot role in a graph which is created in star style. Obviously, triangles and stars are the two basic regular substructures which appear in graphs most frequently. The triangle embodies the transitivity of a graph and the star shows the hubness of a graph. Moreover, graphs containing more triangles are showing different structures and characteristics to graphs that contain more stars. It is very interesting to know which substructure is popular in a graph.

Firstly, the popular substructure reflects the structure feature of the whole graph. Secondly, based on the frequent appearance of a substructure, the graph can be compressed under Minimum Description Length (MDL) principle [170]. Thirdly, the structure information is very helpful for link prediction.

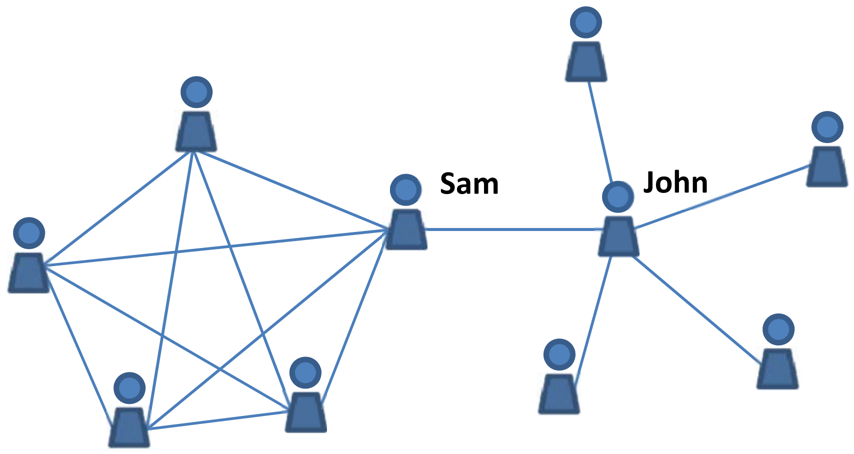


Figure 3.9: Two Differently Structured Sub-graphs.

However, there are some graphs that possess both high transitivity and hubness in different parts of the whole graph. Take the toy graph which is shown in Figure 3.9 as an example. The graph displays the friendship relationship between Sam and John. Supposing that Sam prefers to make friends with local people, and all his local friends are also friends with each other; while John's friends scatters in various countries, so that some of them do not know each other. Therefore, the circle of friends of Sam performs a clique with a large amount of triangles while John is the hub of his circle of friends which is displaying a star. The authors of [154] point out that transitivity increases with the strength of the corresponding community. Traditionally, many algorithms are designed to detect communities which pursue the compactness of inner vertices and sparseness of intra vertices, including spectral clustering [176], Min-Max cuts [58] and others. However, as shown in Figure 3.9, it is meaningful to distinguish among the star-like cluster as John's circle of friends and the clique-like cluster of Sam's friends.

Contributions. In this subchapter,we propose a novel compression-based graph mining algorithm named CXprime (**C**ompression-based e**X**ploiting **Prim**itives). The algorithm is based on exploiting three-node primitives which are the smallest substructures and express both transitivity and hubness of a graph. Unlike complex substructures, three-node primitives are simple and easy to count. Any graph no matter how complex it is can be considered as a combination of three-node primitives. Moreover, we exploit the differences in the relative frequency of three-node primitives for graph compression. Based on the idea of MDL, frequently appearing primitives are effectively compressed in short bitstrings and rarely appearing primitives represented by longer bitstring. Due to the fact that three-node primitives are appropriate for representing both triangular and star substructures, CXprime is designed to distinguish and partition graphs with different substructures. The main contributions of CXprime are summarized as follows:

- **Discovery of graph structure**: CXprime automatically discovers the basic structure type of a graph. Relating data mining to data compression, the core of CXprime is a coding scheme based on three-node primitives allowing to model both triangular and star structures. Emphasizing the characteristics of a triangle graph and star graph separately, CXprime comprises a Triangle Coding Scheme and a Star Coding Scheme. By applying these two coding schemes to an unknown graph we can determine its structure type either as star-like or triangle-like.

- **Graph partitioning based on structures**: Based on the Triangle Coding Scheme and Star Coding Scheme which is proposed by CXprime, the graph can be partitioned into subgraphs with star or triangular structures. Furthermore, the number of clusters can be selected automatically since CXprime is based on the idea of Minimum Description Length.

- **Link prediction based on structures**: CXprime allows us to detect the structure type of a graph. We exploit this information to design a novel unsupervised link prediction score. Experiments demonstrate that this structure-based score outperforms existing techniques which impressively demonstrates that structure information is very useful for graph mining.

The remainder of this subchapter is organized as follows: In the next section, we elaborate our coding scheme. Section 3.2.2 describes the algorithm in detail. Section 3.2.3 shows experiments and results. Related work is discussed in Section 3.2.5.

3.2.1 Graph Compression

Suppose we want to transfer a graph G over a communication channel from a sender to a receiver. We consider an unweighted and undirected graph G with N nodes in this thesis. The graph is provided by its adjacency matrix A with entries $A_{i,j}$ specifying whether there is an edge among the nodes i and j. To transfer G we need to compress each entry in its adjacency matrix A. If we do not have any knowledge on the structure of G, the coding costs are provided by the entropy of A. Since G is an undirected graph, A is symmetric and we only need to encode the upper or lower half of A without the diagonal (representing self-links which are never or always set by convention). Regardless of the type of graph and its characteristics, which can be e.g. scale-free, small-world, Erdos-Renyi, clustered, dense or sparse, we can represent every graph by a bit string of length corresponding to the entropy of its adjacency matrix A. To encode a single entry $A_{i,j}$, we need an average of $H(A)$ bits, where $H(A)$ denotes the entropy and is provided by:

$$-(p(e) \cdot \log_2(p(e)) + p(ne) \cdot \log_2(p(ne))),$$

where $p(e)$ stands for the probability to observe an edge in G, corresponding to the percentage of 1s in A, and $p(ne)$ analogously. Thus the total coding costs are provided by: $N \cdot (N-1)/2 \cdot H(A)$.

If G contains regularities in the form of frequent structure primitives like triangular or star structures, we can compress it more effectively than its entropy. Note that our primary focus is not on compacting the data for transmission over a communication channel but on mining the truly relevant patterns in the data in an unsupervised way. However, there is a direct relationship between data compression and knowledge discovery: The better a set of patterns fit to the data, the better is the compression, i.e. the greater are the savings in coding costs over the entropy which serves as a baseline. In the following section, we elaborate concrete coding schemes including structure primitives which are characteristic for major types of real world graphs, including small-world and scale-free.

Basic Coding Paradigm

Three-node primitives (substructure with three nodes) are the smallest substructures which can embody both connectivity and transitivity of a complex graph. Figure 3.10 enumerates all possible link patterns among three nodes in an undirected graph. In a random graph, each edge exists with the same likelihood and does not depend on the existence of other edges. Therefore, a random graph requires coding costs corresponding to its entropy and cannot be represented more efficiently. If a graph is characterized by transitivity or star-like structures, the existence likelihood of an edge depends on the existence of other edges. In a graph with many star-like hubs, if e.g. node B is already connected to node A (cf. Figure 3.10 (c)), then node C is also connected to A with a high likelihood (cf. Figure 3.10 (g)). In a highly transitive graph, when we know that there are two edges among tree nodes (cf. Figure 3.10 (g)), also the third edge closing the triangle (cf. Figure 3.10 (h)) exists with a high likelihood. In other words, the probability of observing the third edge BC (cf. Figure 3.10 (h)) is very high under the condition that we already observed a potential triangle formed by two edges. We can exploit these typical variations in conditional probabilities to effectively compress structured graphs as follows (see Figure 5.2):

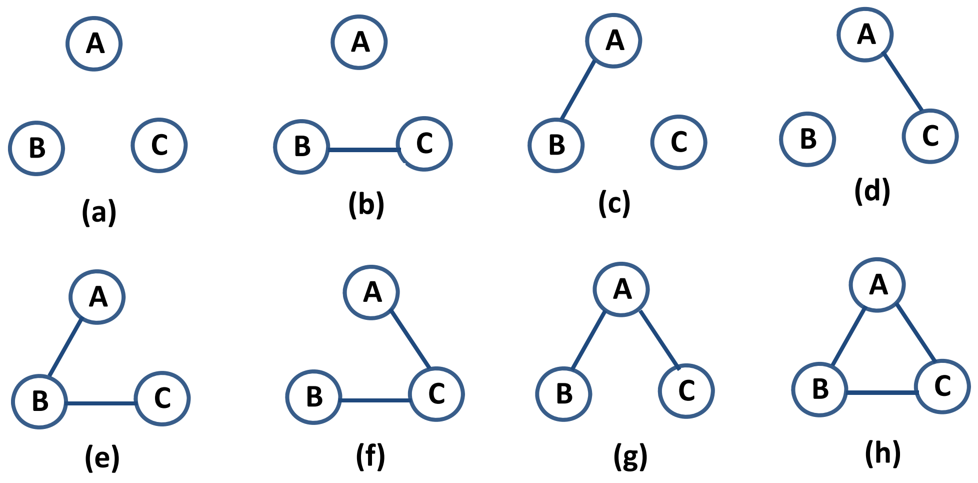


Figure 3.10: All Possible Connections of Three-node Primitives

Fixed Processing Order for Coding and De-Coding. First, the sender and the receiver agree on some fixed order for encoding and de-coding the adjacency matrix A, which can be column-wise, row-wise or diagonal-wise. For an undirected graph, the code is a bitstring composed of $N \cdot (N-1)/2$ codewords. By the fixed coding and de-coding order, the receiver always knows which codeword corresponds to which entry $A_{i,j}$ without any ambiguity or information loss. To encode the graph in Figure 5.2(a), without loss of generality, we select a diagonal-wise order of processing. The diagonal colored in black in Figure 5.2(b) does not need to be transferred, since it represents self-connections which are set by convention. To encode the first off-diagonal, we cannot use any conditional probabilities since each entry corresponds to a single edge among two different nodes and we have no information on three-node primitives.

Case Distinctions based on Conditional Probabilities. Starting from the second off-diagonal, we can exploit conditional probabilities together with case distinctions depending on the information we have already seen before in the processing order. In particular, we define three conditional probabilities which can be obtained from counting the relative frequency of three-node primitives (cf. Figure 3.10), e.g. N_a is the frequency of case (a) :

Definition 21 (Basic Conditional Probabilities)**.**

- **No Edge:** We have not seen any edges in three-node primitives so far.
  $p(e|\textbf{No Edge}) = \frac{N_{bcd}}{N_a+N_{bcd}}$

- **Potential Star:** We have already seen one edge. If another link were added next, we would get a star primitive, therefore we call this primitive with only one edge potential star.
  $p(e|\textbf{Potential Star}) = \frac{N_{efg}}{N_{bcd}+N_{efg}}$

- **Potential Triangle:** We already observed two edges. If another link were added next, we would get a triangle primitive, therefore we call this primitive with two edges potential triangle.
  $p(e|\textbf{Potential Triangle}) = \frac{N_h}{N_h+N_{efg}}$

Where $N_{bcd} = (N_b + N_c + N_d)/3$ and $N_{efg} = (N_e + N_f + N_g)/3$, cf. Figure 3.10. For every condition $\sigma \in \{\textbf{No Edge}, \textbf{Potential Star}, \textbf{Potential Triangle}\}$, the probability that no edge exists is provided by $p(ne|\textbf{Condition } \sigma) = 1 - p(e|\textbf{Condition } \sigma)$. The entropy of each condition is provided by:

$$H(\sigma) = -(p(e|\sigma) \cdot \log_2 p(e|\sigma) + p(ne|\sigma) \cdot \log_2 p(ne|\sigma)). \tag{3.17}$$

where e means there is an edge and no edge for ne.

When encoding a particular entry in the adjacency matrix, we can find all previously seen three-node primitives by inspecting the nodes in the corresponding row and column. In our example (cf. Figure 5.2), we are currently coding the third off-diagonal and now want to encode the red entry corresponding to the edge CG in row 3 and column 7. We already have information about the nodes D, E, and F. We know that D is not connected to C nor G from the entries (3, 4) and (4, 7). We also know that E is connected to C and G ((3, 5) and (5,7)). Finally, we know that F is connected to G but not to C ((6,7) and (3,6)). We observed all three conditions above in the previously seen data. Which one should we use to encode the current entry? Since we aim at compressing the data as much as possible we always select that condition which is expected to give us the shortest codeword. This is the condition having the lowest entropy. We could also see our three conditions as alternative classifiers predicting the current entry. If we have the choice among several classifiers, i.e. if multiple conditions apply, it makes sense to select the classifier which is most certain about the current case. For our example graph we have $p(e|\textbf{Potential Triangle}) = 0.72$, $p(e|\textbf{Potential Star}) = 0.35$ and $p(e|\textbf{No Edge}) = 0.68$. Thus, we select the condition **Potential Triangle**, since it allows us to encode the current entry with 0.86 bits in average, while **Potential Star** would require 0.93 bits and **No Edge** 0.9 bits. The current entry is an edge, so we use the codeword representing an edge under the condition **Potential Triangle** to encode it. This coding scheme is de-codeable without information loss since for coding and de-coding sender and receiver perform the same case distinction based on the same data.

Parameter Costs. To decode the bitstring, the sender needs the codebook consisting of the conditional probabilities required to perform the case distinction and the code table saying which bitstring represents an edge or no edge in every case. Following [169], these parameter costs can be approximated by Eq. 3.18:

$$CC_{param} = 0.5 \cdot num \cdot \log_2 \frac{N \cdot (N-1)}{2}, \tag{3.18}$$

where N is the number of nodes in the graph and num denotes the number of parameters, which is three in our case since we consider three different conditional probabilities.

Efficient Implementation with Adjacency Lists. For efficient coding and de-coding the sender and receiver use adjacency lists. Every time a new entry is processed, it is inserted into the adjacency lists of both corresponding nodes. Figure 5.2(c) displays a snapshot of the adjacency lists before processing CG. In order to collect the applicable cases for encoding or de-coding CG, instead of looking into the corresponding row and column of the adjacency matrix, we scan the adjacency lists of nodes C and G. In particular, we start with the list of G from the beginning and with that of C from the end. In the first step, we retrieve F as the first node in the list of G and E as the last node in the list of C. We know that F is adjacent to G but not to E, which means that the condition **Potential Star** is applicable. Having processed F, we move one step forwards in the list of G and detect the matching node E, from which we can deduce that **Potential Triangle** also is applicable. Having processed E, we can move one step in both lists, which means that we come to the end of the list of G and obtain B in the list of C. Due to the processing order, information on three-node primitives formed with node B is not yet available, therefore we can stop as soon as we detect a node coming before C. But we know that we already have information on D. Therefore, we detect that condition **No Edge** is also applicable.

Extended Coding Paradigm

The basic coding paradigm only considers the primitives with three nodes, which is the simplest substructure in a graph. During the diagonal-wise coding process, we can see less previous information in the beginning and more at the end. Therefore if primitives with more nodes can be used, we could compress the graph more effectively and get more knowledge from it. However, counting probabilities of primitives with more nodes would require high computational costs.

We choose some primitives with more nodes to extend the basic coding paradigm as Figure 3.12 shows. These primitives are frequent in real world graphs, like dense communities or hub nodes and their neighbors.

Definition 22 (Higher-order Conditional Probabilities).

- **Multiple Triangles**: There are multiple points connecting both B and C, which means previously we can see multiple dual-connected edges. As shown in Figure 3.12 (a) and (b);

- **Strong Star**: There are multiple points connecting B or C, which means previously we can see multiple single-connected edges. As shown in Figure 3.12 (c) and (d).

Suppose the high-order conditional are represented as **Condition** $\sigma' \in$ **{Multiple Triangles, Strong Star}**. The probabilities of existence an edge e under the condition σ' are provided by:

$$p(e|k \cdot \textbf{Condition } \sigma') = \frac{\sum_{i=1}^{E} \mathcal{C}_{m_i}^{k}}{\sum_{i=1}^{E} \mathcal{C}_{m_i}^{k} + \sum_{j=1}^{NE} \mathcal{C}_{n_j}^{k}} \tag{3.19}$$

Where E is the number of edges in a graph G, and NE is the number of pairs of nodes without connection. m_i with $i = 1, 2, ..., E$ is frequencies of **Condition** σ' for the connected entry, and n_j with $j = 1, 2, ..., NE$ is frequencies of **Condition** σ' for the unconnected entry. $\mathcal{C}$ is combination symbol. k is the actual frequency we can see for each entry.

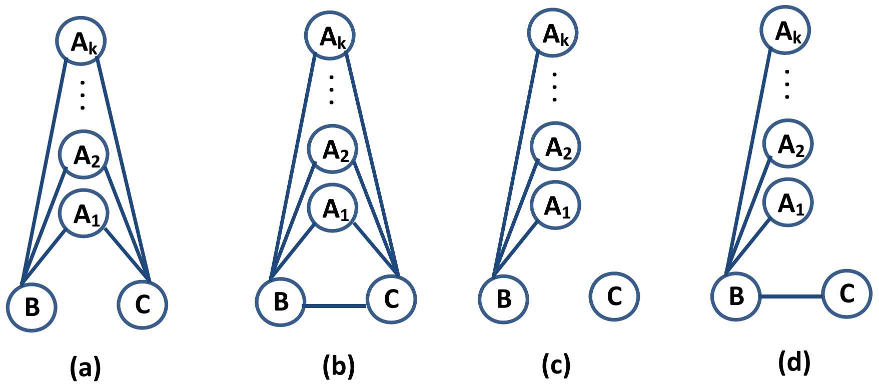


Figure 3.12: Multiple Primitives.

No extra effort is required to obtain these higher-order conditional probabilities, since they are calculated without counting the number of edges (again), but are created by calculating the combination of pair of nodes that have the same basic primitives, e.g. two linked nodes share n neighbors then they have C_n^k triangles. We use Eq. 3.19 to calculate the statistic for multiple triangles and strong stars. In our experiments of both synthetic and real data sets, the probabilities in Eq. 3.19with different k become stable when k increases. Therefore, we only compute the probabilities when $k \leq 4$ in this thesis.

To be able to automatically detect the structure type of a graph we now introduce a *Star Coding Scheme* and a *Triangle Coding Scheme*.

The Star Coding Scheme employs for coding the following set $\mathcal{S}$ of conditional probabilities $\mathcal{S}$:= {**No Edge**, **Potential Triangle**, **Potential Star**, **Strong Star**} while the Triangle Coding Scheme works with a different set $\mathcal{T}$:= {**No Edge**, **Potential Star**, **Potential Triangle**, **Multiple Triangles**} as specified in the basic and higher-order conditional probabilities, cf. Definitions 21 and 22. Upon encoding or decoding each entry, the sender or receiver always select that conditional probability of $\mathcal{S}$ and $\mathcal{T}$ having the lowest entropy respectively. The overall coding cost CC_S for the Star Coding Scheme is provided by:

$$CC_S = \sum_{e} \underset{\min H(\mathcal{S})}{L(e|\mathcal{S})} + \sum_{ne} \underset{\min H(\mathcal{S})}{L(ne|\mathcal{S})} + CC_{param} \tag{3.20}$$

where $L(e|S) = -\log_2 p(e|\mathcal{S})$ is the coding length of a connected entry, $L(ne|S) = -\log_2 p(ne|\mathcal{S})$ is the coding length of a unconnected entry, CC_{param} represents the parameter costs as specified in Eq. 3.18.However, we now have $3+k-1$ parameters to consider. Three of them are basic probabilities, while $k-1$ of them are higher order probabilities except for corresponding basic three-node primitive. The coding cost CC_T of the Triangle Coding Scheme is determined analogously applying the corresponding set $\mathcal{T}$.

Both coding schemes contain all basic conditional probabilities since we need to be able to represent any possible link pattern among three-node primitives with both schemes. The higher-order probabilities emphasize and reward star and triangle structures by assigning very short bitstrings to them. The coding and de-coding process with these two extended coding schemes works as explained in Section 3.2.1. Also here, to encode the first diagonal of A, we use the entropy provided by the general edge existence probability, which is omitted in the above definition for clarity of presentation.

3.2.2 Algorithm: CXprime

In this section, we present our algorithm CXprime to mine useful information considering the underlying structure primitives from graphs. The core part of CXprime are the two coding schemes, Triangle and Star Coding Scheme, which are proposed in the previous section. CXprime is able to distinguish graphs with high transitivity from graphs with a high amount of hubness by comparing the coding cost of both coding schemes. Apart from distinguishing the graph structures, we combine the coding scheme with K-means for graph partitioning and propose a new link prediction technique exploiting graph structure information.

Graph Partitioning

Considering the complexity of a graph structure, it can be partly in a high transitivity state and partly consisting of many hubs. Traditionally, dense communities with high transitivity are formed by several triangular structures. However, communities with hub are playing a pivot role in graphs as well. Combining the two proposed coding schemes regarding the different structures in a clustering process we introduce a novel idea for graph clustering: Partitioning by structure. The K-means like clustering of CXprime is guided by the proposed coding schemes, which partitions graphs into parts based on their respective structural properties that compress the whole graph best. To avoid overfitting, we follow the principle of Minimum Description Length and require that not only the data but also the structure primitives considered in the codebook need to be encoded.

Based on MDL, we extend the proposed Triangle and Star Coding Scheme to be used in clustering. More specifically, the MDL principle is applied to compress a set of candidate clustering models, where in our case different models correspond to different partitions. We use our Triangle and Star Coding Scheme to compress the clusters to test whether the subgraph of an original graph contains triangular or star structures. The coding cost for graph G under the clustering model M with K clusters $\{G_1, G_2, ..., G_K\}$ is provided by :

$$L(G|M) = \sum_{i=1}^{K}(\min(CC_S(G_i), CC_T(G_i))) + CC_B \tag{3.21}$$

where CC_S represents the coding cost of the star coding scheme, CC_T is the coding cost of the triangle coding scheme and CC_B indicates the costs describing the edges between different graph clusters, for which simple entropy coding is used.

To avoid overfitting, MDL not only include the cost for coding the data with the model $L(G|M)$ but also the cost $L(M)$ for the model. We need to specify the clustering assignment and conditional probabilities of the Triangle or Star Coding Schemes for each cluster.

$$L(M) = \sum_{i=1}^{K} |G_i| \log_2(\frac{N}{|G_i|}) + \sum_{i=1}^{K} CC_{param}(G_i) \tag{3.22}$$

where the first term represents the coding cost for the clustering assignment and the second term represents parameters cost.

Finally, the total coding cost for the whole data set with the clustering model M can be obtained by:

$$L(G, M) = L(M) + L(G|M)$$

(3.23)

The MDL based clustering algorithm is depicted in Algorithm 5. During initialization, we choose K nodes with the longest shortest path between each other as cluster centers. Then neighbors of the center nodes are directly assigned to corresponding clusters. If remaining nodes are neighbors of nodes in a cluster, they will be assigned to this cluster as well. Finally, all nodes are roughly assigned to K clusters and we calculate the star coding cost CC_S and triangle coding cost CC_T of these clusters, then choose the minimum value as initial coding cost. In the iteration phase, each node is moved to all the other clusters to test whether it reduces the coding cost. If the new coding cost is decreased, the new graph clusters will be kept. Otherwise, the nodes will be moved back to their former cluster. The iteration terminates when the clustering labels do not change. Due to the fact that CXprime is a MDL-based algorithm, the number of clusters K can be chosen automatically by searching the minimum coding cost without using any parameter.

Link Prediction

In highly transitive graphs as dense communities, triangular substructures appear the most frequently. It implies that if two nodes are involved in more **Potential Triangle** with other nodes, there will be a higher probability that the two nodes will be linked. On the other hand, star substructures are the most common patterns

Algorithm 5. Graph Partitioning

```
Input: Graph $G$
Output: Graph Clusters $G_c = G_1, G_2, ..., G_K$
 1: Select $K$ nodes as cluster centers, and assign nodes to $K$ clusters;
 2: Calculate coding cost $CC_{old}$
 3: while Converge do
 4:    Reassign nodes to other clusters;
 5:    Recalculate coding cost
 6:    if $CC_{new} > CC_{old}$ then
 7:       Move nodes back;
 8:    end if
 9: end while
10: return $G_c$.
```

appearing in a graph with several high degree hubs. Thus if two nodes are involved in more **Potential Star** with other nodes, there is a higher probability that the two nodes will be connected in a star-like graph. Benefiting from structures of graph detected by CXprime, we propose a new unsupervised link prediction method. Specifically, we combine the two situations and give a new prediction score which is shown as:

$$S_{CXprime}(e) = \frac{CC_T}{CC_T + CC_S} \cdot f_T(e) + \frac{CC_S}{CC_T + CC_S} \cdot f_S(e),$$

(3.24)

where CC_T and CC_S are the coding cost of Triangle Coding Scheme and Star Coding Scheme respectively, e is the edge that will be predicted, f_T is the frequency of **Potential Triangle** after normalization and f_S is the number of **Potential Star** after normalization. These frequencies can be used for link prediction after we give weights to them based on the graph type. Coding costs of a given graph on both triangle and star type are adopted to generate weights. If $CC_T > CC_S$, then the graph contains more triangular structures than star structures, and the frequency of triangle f_T will be assigned bigger weight. If $CC_T < CC_S$, star structures dominate the graph, thus we give f_S a bigger weight.

Runtime complexity. The runtime complexity of CXprime to compress a graph G with N nodes and E edges involves: calculating the statistics of structure primitives and coding the adjacency matrix. Gathering the statistics with the adjacency list we need to go through each pair of vertices $O(N \cdot (N-1)/2)$ and compare their neighbors $O(2E/N)$, where E/N is the average number of edges for each vertex. The asymptotic complexity for this task hence reduce to $O(N \cdot E)$. Similarly, the complexity of the coding part is $O(\sqrt{N \cdot E} \cdot N)$. For graph partitioning, we use an efficient K-means-style approach which is linear in N and usually converges very fast.

3.2.3 Experimental Evaluation

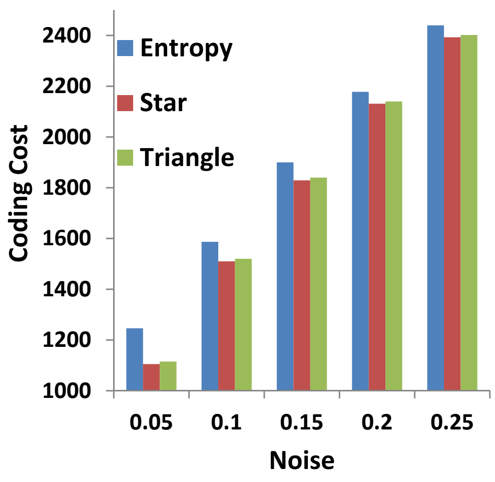
This section evaluates the three major contributions CXprime separately. CXprime is implemented in Java. All experiments have been performed on a workstation with 2 Duo 2.4GHz CPU and 4.0 GB RAM.

Discovering the Graph Structure

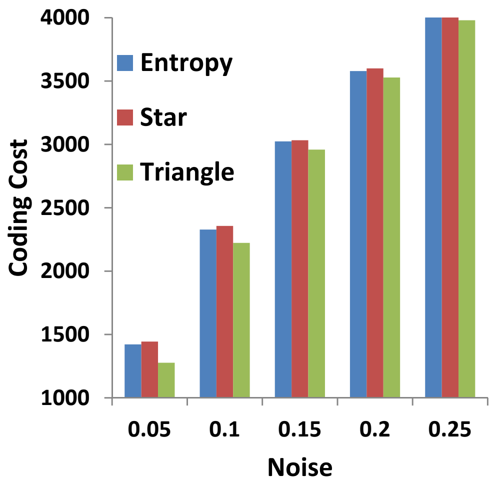
With the two different coding schemes for stars and triangles, CXprime is able to identify whether the graph is formed by star structures or by triangular structures. It holds that the coding scheme that shows the minimal coding cost for a given graph indicates which structure appearing more frequently in it. We evaluated the two coding schemes of the algorithm on both synthetic data sets and real data sets, the real data sets are coming from sports and media industry. To prove the efficiency of the compression, the acquired coding cost is compared with the entropy of the graph.

Synthetic Data Sets. We generate two types of graphs which clearly show the differences between star and triangular structures. One type is mainly constructed by star-like structure primitives that we call it *star graph*. The other graph is mainly composed of triangular structure primitives which is named *triangle graph*. The number of nodes in both cases is fixed to 100. Specifically, we generate a star-like graph with three hubs, based on which three single star structures with equally same number of nodes are formed. Under the condition of keeping the original star-like structure, noise edges are added to generated graph. The percentage of noise increases from 0.05 to 0.25. Obviously, a clique contains numerous triangular structures. We generate triangle graph based on one clique structure. Similarly, in order to keep the original triangular structure, we remove edges from the generated graph with the percentage of removed edges increasing from 0.05 to 0.25.

As shown in the bar charts of Figure 3.13a and 3.13b, our two coding schemes of CXprime are evaluated on each generated star graph and triangle graph with the percentage of disturbing edges increasing from 0.05 to 0.25 separately. While the entropy of the graph serves as a baseline. Figure 3.13a illustrates that the Star Coding Scheme has a lower coding cost than the Triangle Coding Scheme in star graph, which demonstrates that CXprime successfully detects that star structure frequently appears in all cases. Analogously, in Figure 3.13b the Triangle Coding Scheme has a lower coding cost in the triangle graph, which proves that the frequent appearance structure is triangular structure.



(a) Star Graph.



(b) Triangle Graph.

Figure 3.13: Coding scheme evaluation on synthetic data.

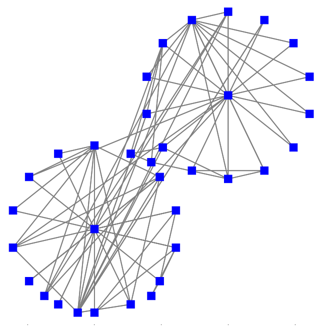
Real Data Sets[2] Our first real world data set called “Zachary’s karate club” [204] is a social network with 34 nodes which demonstrates the relationship between members of a karate sports club at a US university in the 1970s. The other real world data set “Les Misérables” [118] is a network with 77 nodes indicating characters in Victor Hugo’s famous novel of the same name. The edges are representing the connection between any pair of characters which appear in the same chapter of the book. Both data sets are obtained from the UCI Network Data Repository. Table 3.3 shows the coding costs of “Zachary’s karate club” and “Les Misérables” which calculated by entropy, Star Coding Scheme and Triangle Coding Scheme, respectively. In the case of “Zachary’s karate club” the Star Coding Scheme obtains

[2]http://networkdata.ics.uci.edu/index.php

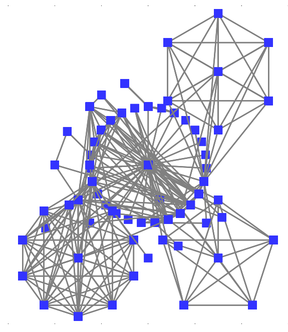
the minimum coding cost, which implies that this graph is more star-like. In order to further evaluate results of distinguishing graph substructures, we visualize "Zachary's karate club" data in Figure 3.14a. Seen from the figure, there are two striking hubs inside the graph. In terms of "Les Misérables", comparing with other coding schemes, the Triangle Coding Scheme yields the smallest value, which indicates that there are more triangle structures than star structures in the graph. Moreover, Figure 3.14b shows that there are a large amount of obvious triangular structures in the "Les Misérables" network.

Table 3.3: Distinguishing Graph Structure on Real Data Sets

	Entropy	Star	Triangle
Zachary's karate club	330.9	**325.6**	331.3
Les Miserables	1251.4	986.6	**895.1**



(a) Structure of Zachary's karate club (star-like).



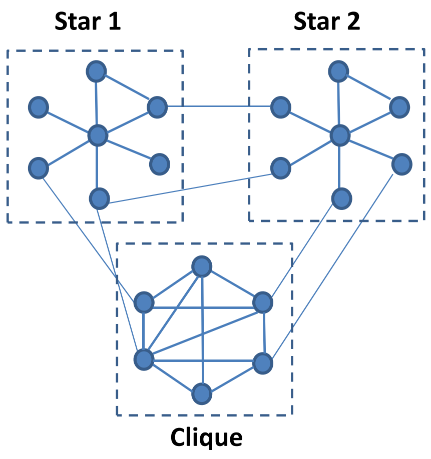
(b) Structure of Les Miserables (triangle-like).

Figure 3.14: Coding scheme evaluation on real world data.

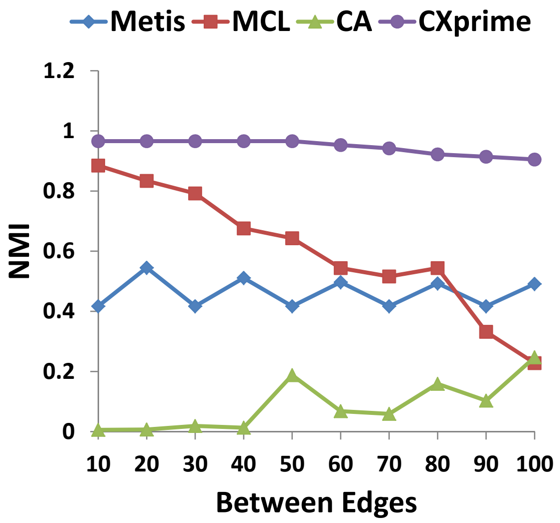
Graph Partitioning

In this section, we compare CXprime with classical graph partitioning algorithms, such as Metis [113] and Markov Clustering (abbreviated as MCL) [187]. Besides, we compare CXprime with cross-association (abbreviated as CA) [41], which is also a compression-based graph clustering algorithm. Metis and MCL require input parameters. For MCL, we used the default parametrization. CXprime and CA automatically find clusters without any parameter due to their information-theoretic approaches.

Synthetic Data Sets. We generate two synthetic data sets with different structural clusters, and evaluate the graph partitioning performance of CXprime and the comparison methods. We generate each type of data set 10 times using a different random number generator and output the average results.



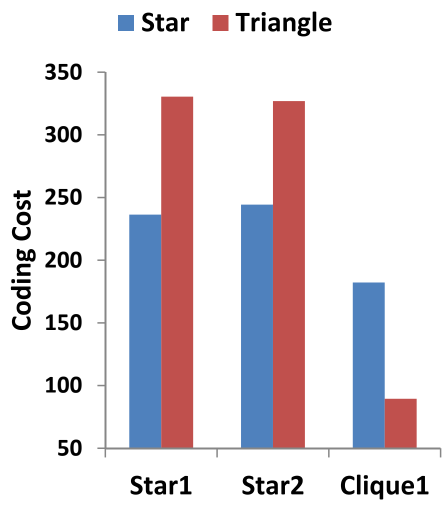
(a) Sketch Figure



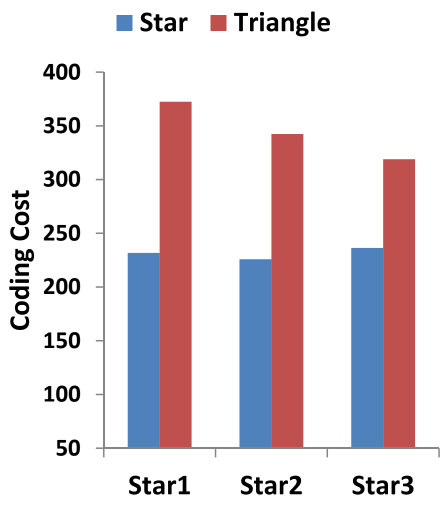
(b) Partitioning Results

Figure 3.15: Syn1 with Two Stars and One Clique.

The first synthetic data set *Syn*1 is composed of two star clusters and one clique cluster (100 nodes each) with sketch map shown in Figure 3.15a. The star cluster is generated with one hub connecting all the other nodes, besides 20 edges are added to it as noise. The clique cluster is created by making a full connected graph first and then 20 edges are removed from it. The edges between each pair of two clusters are randomly selected to connect them, which is why we altered their number from 10 to 100 to evaluate their affects on graph partitioning. Since we know the class label of each node as ground truth, the Normalized Mutual Information (NMI) [189] is used to evaluate the clustering results, and NMI scales between 0 and 1, where 1 means a perfect clustering and 0 means no agreement at all among class and cluster labels. Figure 3.15b shows curves of NMI values when implementing CXprime and comparison algorithms on $Syn1$ with different number of between edges. Benefiting from finding the structure of the star cluster, CXprime clearly performs better than the other methods with a NMI above 0.9 even when there are 100 edges between each pair of two clusters. Figure 3.16a depicts the coding costs of detected clusters in Star Coding Scheme and Triangle Coding Scheme. As expected, Star Coding Scheme gives less bits for two star-like clusters, while Triangle Coding Scheme gives less bits for the clique-like cluster. Metis performs good when there are less edges (below 30) between two clusters, however its performance severely degrades when there are more edges in between (below 0.3 when there are 100 between edges). MCL cannot find correct clusters with a NMI about 0.5 for all the cases. CA fails to detect correct clusters with a NMI no bigger than 0.3, because there is no dense region in a star cluster. Interestingly, CA gets better results when there are more edges in between, which shows that CA can not find sparse clusters.



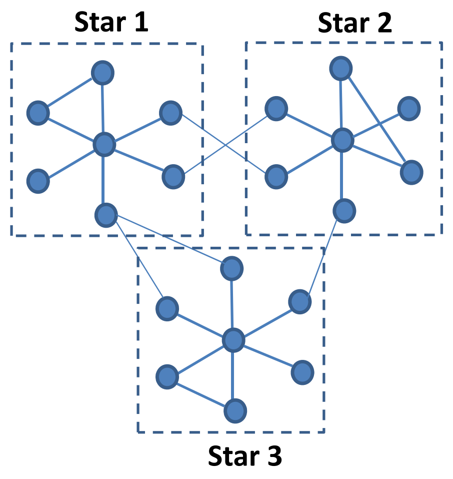
(a) Syn1



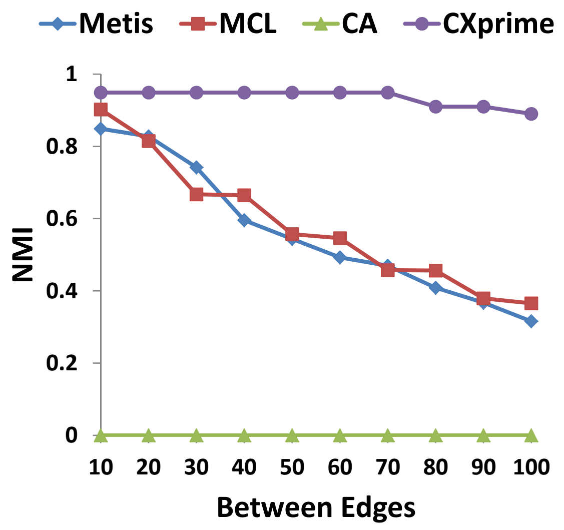
(b) Syn2

Figure 3.16: Coding Cost of Clusters.

The second synthetic data set $Syn2$ is composed of three star clusters and each star contains 100 nodes, the sketch map is shown in Figure 3.17a. Similarly, 20 edges are added to each star cluster as noise in order to keep the star structure. Each pair of star clusters is connected with randomly selected edges with their number ranging from 10 to 100. The clustering results of *Syn*2 which are evaluated by NMI are depicted as curve graph in Figure 3.17b. Seen from the figure, CXprime clearly performs better than the other methods with a NMI above 0.9 even when there are 100 edges between each pair of two clusters. Figure 3.16b depicts the coding costs of detected clusters in Star Coding Scheme and Triangle Coding Scheme, in which the Star Coding Scheme compress all three star-like clusters with less bits. Metis and MCL perform good when there are less edges between clusters, however their performances severely degrades when there are more edges in between. CA cannot detect any clusters in this data set, because CA can not find star-like structures.



(a) Sketch Figure



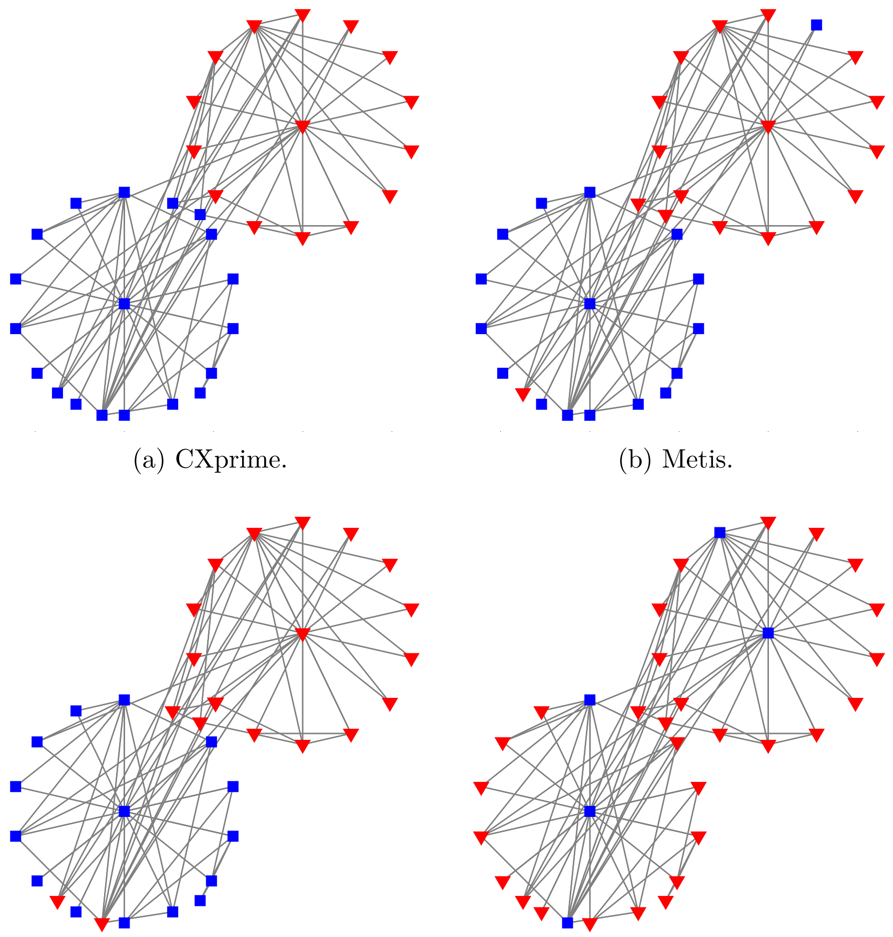
(b) Partitioning Results

Figure 3.17: Syn2 with Three Stars.

In summary, the results of CXprime are clearly superior than those of the comparison partners on all synthetic data. This demonstrates that CXprime is suitable to detect clusters according to their structure type.

Real Data Sets. Two real data sets are used to evaluate the performance of CXprime on graph partitioning, “Zachary’s karate club” graph and a subgraph of the DBLP co-authorship graph.

"Zachary's karate club" graph is small and therefore we can asses its structure by directly drawing it by splitting hub nodes as shown in Figure 3.14a, which helps us to interpret the partitioning results. CXprime and Cross Association automatically detect two clusters, MCL finds two clusters under the default parameter setting as well. Therefore, we set the cluster number to 2 for Metis. Since class labels are unavailable, we visualize the graph for evaluation. The graph partitioning results are depicted in Figure 3.18, in which different symbols stand for different cluster labels. Seen from Figure 3.18a, two clusters are detected by CXprime, the one labeled with red triangle is sparser and the one labeled with blue square is denser. Metis and MCL perform well on this data set (Figure 3.18b and 3.18c), but still have several points grouped differently. However, note that the results of Metis and MCL only consist of the clustering without giving any information on the cluster content. CXprime is the only method providing us not only the clustering but also the interesting information that the content of one clusters is dominated with star structure and the other one contains more triangle due to the dense connection. Cross Association completely fails to detect clusters in such data set as Figure 3.18d shows, because there is no very dense region in this graph.


(a) CXprime. (b) Metis.

(c) MCL. (d) Cross Association.

Figure 3.18: Graph Partitioning of Zachary's karate club Data.

The DBLP[3] network contains information on which researchers are publishing together and how each research group evolves over time. It has the advantage that we can interpret the results relatively easy based on our personal knowledge and on the knowledge provided open source by DBLP even though the data is unlabeled. We generated our test data set by taking all co-authors of three well-known international professors, namely "Jiawei Han", "Christos Faloutsos" and "Hans-Peter Kriegel" as nodes and expect the professors to be the nodes with the highest degree (hubs). The co-authors and professors are connected if any two of them cooperated on a chapter together. The data set consists of 1014 vertices with

[3]http://dblp.uni-trier.de/

Table 3.4: Example of differing people in MCL and CXprime

	MCL	CXprime
Hui Zhang	han-group	**faloutsos-group**
Padhraic Smyth	han-group	**faloutsos-group**
Wei-Ying Ma	han-group	**kriegel-group**
Senqiang Zhou	**han-group**	faloutsos-group
John Paul Sondag	**han-group**	faloutsos-group

5828 edges between them. In the following we will refer to the three clusters that we expect our comparison methods to find as “han-group”, “faloutsos-group” and “kriegel-group”. Therefore, we set the number of cluster to 3 in Metis, MCL was working with its default parameter and CXprime and CA are both parameter-free algorithms.

The extracted DBLP subgraph does not connect densely, thus CA is not able to find any meaningful clusters. Metis finds three cluster, but one of the clusters contains two hubs “Christos Faloutsos” and “Hans-Peter Kriegel”, which considerably deviates from the ground truth. The result of MCL and CXprime is similar, all three professors are as expected in different clusters, and in the overall look the quality of both results are good. In detail, they differ in the han-group in 26 people, in the faloutsos-group in 27 people and in the kriegel-group only 8 people. Some examples are shown in Table 3.4. Among these different peoples both methods are not totally correct and most of the differences exist between han-group and faloutsos-group. For example, Hui Zhang and Padhraic Smyth who were falsely put into han-group by MCL. Specifically, Checking from DBLP website, Hui zhang has published two chapters with Prof. Faloutsos but has no collaboration with Prof. Han. And Padhraic Smyth also has only collaborated with Prof. Faloutsos. Moreover, Wei-Ying Ma has published one chapter with Prof. Kriegel, but is falsely put into han-group by MCL. However, both Senqiang Zhou and John Paul Sondag have published one chapter with Prof. Han respectively. But they are falsely grouped into faloutsos-group by CXprime. Therefore, we consider the quality of the results of MCL and CXprime for this dataset as approximately equal.

However, CXprime is the only method detecting the structure of these three clusters. Here, we expect PhDs working at university with the professor to publish in more of a clique structure and external as well as cooperation partners as more of a star one. Also, this evolves over time. CXprime provides the content information of these three clusters that han-group is the biggest group and is more triangle-like. The kriegel-group as the smallest cluster is denser and therefore is formed as triangle-like. And the faloutsos-group is referred to as a more star-like motif, which may caused by containing external students.

Compression Rates. The basic idea of MDL is that the more you compress the data the more you learn from it. Therefore, it is non-trivial to compare CXprime with compression-based methods in terms of compression rates. We compare the compression rates between these methods: our Star Coding Scheme and Triangle coding scheme, our CXprime partition algorithm, and two existing compression-based graph mining algorithms SLASHBURN[109] and cross-association (CA), and entropy is given as a base line. The results for both synthetic ($Syn1$ and $Syn2$ with 10 edges between each pair of clusters) and real datasets are depicted in Table 3.5, which are shown in bits. As only half of the adjacency matrix is considered in this chapter, the compression rates of SLASHBURN and CA are also calculated from the half of the matrix. And the number in bracket are the sizes of blocks for SLASHBURN, we try different settings and output the best compression rate. It is clear that Cxprime outperforms the other methods by achieving higher compression rates in both synthetic and real data sets.

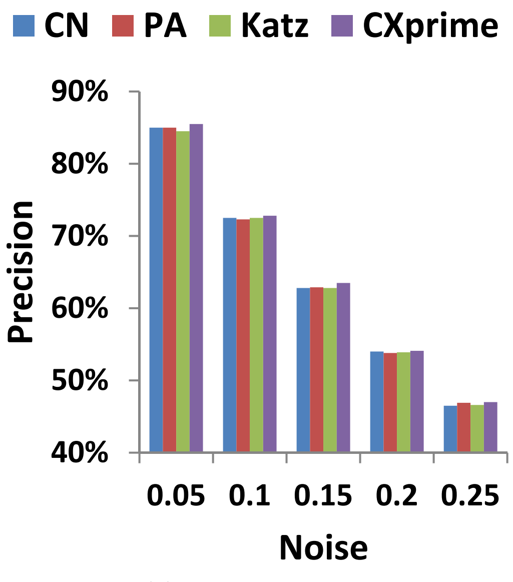
Table 3.5: Compression Rates (Bits)

	Entropy	Star	Triangle	Cxprime	SLASHBURN	CA
Syn1	23215	12906	4345	**1397.28**	4056.6(20)	2230.5
Syn2	3114	1926.8	1930.5	**1390.9**	2399.1(20)	1695.5
Karate	330.9	325.6	331.3	**297.1**	306.6(11)	317
DBLP	46027	44743	39249	**35297**	38538(50)	38485

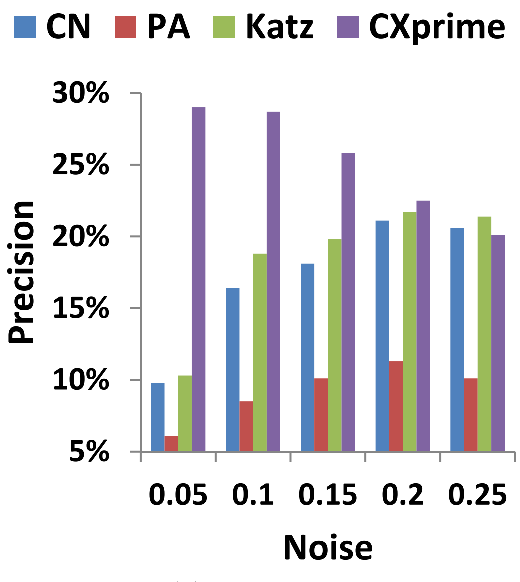
3.2.4 Link Prediction

Considering the structures which can be distinguished by CXprime, an unsupervised link prediction score is proposed. In this chapter, we compare our proposed score with other unsupervised link prediction scores, Common Neighbors (CN), Preference Attachment (PA) and Katz ($\beta = 0.005$). All scores are experimented on both synthetic and real data sets. In order to evaluate the efficiency of our scores, we randomly sample 30% of edges as predicting edges S and deleted them from the original graph. In the resulting graph, we calculate the link-prediction scores of every pair of unconnected nodes and sort them descending. The first $|S|$ edges of each score are selected separately as a predicted result which is expressed as P. After comparing the predicting edges S with predicted edges P, we use the precision $|S \bigcap P|/|S|$ to evaluate the results. All results are the average values of 10 times running the algorithms.

Synthetic Data Sets. We implement the link prediction scores on both triangle graph and star graph which are generated in the same way like the synthetic data sets in section 3.2.3. The precisions of four unsupervised link prediction scores of the triangle graph and star graph with percentage of noise edges ranging from 0.05 to 0.25 are shown in Figure 3.19a and Figure3.19b separately. Clearly, Figure 3.19a shows that CXprime possesses more or less higher precisioncolumn in a triangle graph than the other three scores in each cases. Moreover, seen from Figure3.19b, CXprime occupies the highest position of the four scores in first four cases of star graph. Especially when the noise density is small, the advantage of CXprime is more remarkable.



(a) Triangle Graph.



(b) Star Graph.

Figure 3.19: Precision of Synthetic Data Sets.

Real Data Sets.[4] "Zachary's karate club" and "Copperfield Word Adjacencies"[152] are adopted in this part. "Copperfield Word Adjacencies" is the network with 112 nodes and 425 edges which represent common adjective and noun adjacencies for the novel"David Copperfield" by Charles Dickens. The link prediction precision of each score is displayed in Table3.6 which shows that our proposed score of CXprime outperforms the other three scores of the given comparison methods on these two real data sets.

Table 3.6: Precision of Real Data Sets (%)

	CN	Katz	PA	CXprime
Zachary's karate club	16.9	15.6	16.5	**26.1**
Copperfield Word Adjacencies	11.1	16.5	11.5	**16.7**

3.2.5 Related Work

We briefly survey related work on four relevant topics addressed by CXPrime: graph structure and pattern mining, compression-based graph mining, graph partitioning and link prediction.

Graph Structure and Pattern Mining. Research on the structure of complex networks has already gained significant attention. Most real world graphs follow power-law degree distributions, which can distinguish between an actual real-world graph and any artificial one [40]. In a power-law graph, most vertices have a very low degree, while few ones have extremely high degrees (we call this pattern a star). The existence of these hubs makes the community detection in these real world graphs very difficult, because most existing graph clustering techniques focus on densely connected communities (triangle-types). On the other hand, there are many frequent subgraph mining or motif detection algorithms, [197][175] to mention a few, which aim to find recurrent and statistically significant sub-graphs. Our technique CXprime combines these two approaches and exploits structure primitives to discover the graph structure itself - a novel approach compared to previous work.

Compression-based Graph Mining. Due to the increasing scale of graph data, there are many algorithms proposed for efficiently compressing an graph,

[4]http://networkdata.ics.uci.edu/index.php

e.g.[32][45] to mention a few. However, the compression aspect is not the major focus in this paper, but knowledge discovery from an information-theoretic background. SUBDUE [93] and cross-association [41] are two famous algorithms also relying on the Minimum Description Length principle to identify patterns in a graph. SUBDUE uses a heuristic search guided by MDL to find specific patterns minimizing the description length of the entire graph. Other than CXprime, SUBDUE is not able to find the global general structure of a graph. Cross-association is another co-clustering method, which can be applied for unipartite graphs as well, processing a binary matrix and seeking clusters of rows and columns. Then the matrix is divided into homogeneous rectangles which are representing the underlying structure of the data. Cross-association can only find dense triangle communities and is therefore not suited for many sparse real graphs, while CXprime is explicitly designed for such types of graphs. Another, more recent work called SLASHBURN [109] was proposed for graph compression exploiting the hubs and the neighbors of hubs. SLASHBURN uses the power-law characteristic for compression, and can only exploit dense communities.

Graph Partitioning. There are plenty of works on graph clustering or graph partitioning, including spectral clustering [176], Min-Max-Cut algorithms[58], Metis [113], Markov Clustering [187] for unipartite graphs, co-clustering [57], cross-association [41] and SCMiner [69] for bipartite ones. All these methods aim to find regions with more intra edges than inter edges, in which densely connected subgraph communities can be found. However, none of them considers interesting sparse patterns like stars, which are prevalent in real world graphs. Therefore, these approaches fail in detecting star-like communities, while the proposed CXprime can distinguish between star-like and triangle-like sub-structures.

Link prediction. Inferring whether two disconnected nodes will be linked in the future, based on available graph information is the task of link prediction in graph mining. Liben-Nowell and Kleinberg [127] summarize some unsupervised link prediction approaches for social networks. For example, common neighbour, preferential attachment [151], Katz [114] and others. Specifically, preferential attachment is based on the idea that two nodes with higher degrees have a higher probability to be connected. Katz defines a score that sums up the number of all paths between two nodes where short paths are weighted stronger. Obviously, common neighbour and Katz are more effective in triangle graphs, while preferential attachment works better in star-like graphs. However, none of the previous methods consider the underlying graph structure information which is important for link prediction. To the best of our knowledge, CXprime is the first method that is using structure information to improve the quality of link prediction.

3.3 Conclusion

In this chapter, we introduced CXprime and IROC, CXprime being an multi-functional algorithm for mining graphs based on structure primitives and IROC a non-redundant method for finding overlapping communities on attributed graphs. The two key ideas of CXPrime are to model the transitivity and the hubness of a graph using three-node primitives and to exploit the relative frequency of these structure primitives for data compression and knowledge discovery. We demonstrate that the combination of these two ideas is very useful for (1) automatically detecting the structure type of a graph as star-like/scale-free or clique-like/small-world, (2) clustering the graph into homogeneously structured sub-graphs and (3) accurate link prediction. Our experiments demonstrate that the knowledge about the structure type is very interesting for interpreting graphs and that our novel structure-based cluster notion is a valuable complement to traditional graph clustering methods searching for dense sub-networks only. Regarding link-prediction, we outperform existing methods especially on star-like graphs which demonstrates that the knowledge about the structure type is indeed very useful for graph mining.

Our method IROC applied the concept of information theoretic measures using Minimum Description Length to assess and optimize the quality of a multi-mode attribute graph clustering. This approach allows to retrieve high quality clusterings without requiring the user to specify any parameters and redundancy of clusters in both the cluster topology as well as the selected attributes. While IROC's performance does not scale to large networks we apply tensor factorization to our information theoretic approach to gain a balanced result considering runtime and cluster quality. Our experiments showed that IROC and its optimization TF-IROC are able to outperform state-of-the-art methods on synthetic data. For large real social network data, IROC and TF-IROC yield the highest accuracy in terms of finding labelled ground-truth clusters. Next, we hope to aquire large multi-mode social network datasets, where connections between the same users are recorded for different social networks. In this case, which IROC and TF-IROC are designed for, we expect even more convincing experimental results.

CHAPTER 4

Methods for Spatio-Temporal Data

Parts of this chapter have previously been published in PAKDD 2017 [98].

In this chapter we deal with timeseries on environmental spatio-temporal data. This data is by far the largest in volume (up to 70 TB). For such sensoric data we create a framework consisting of three phases: First, outlier analysis on environmental timeseries taking the direct vicinity into account and creating an environmental extremeness measure as pruning criteria for outlier. Second using such outlier for tensor sparsification and reducing dimensionality with that factorization. Lastly, classifying and predicting such outlier, natural hazards and in this case tropical storms, with a variety of classifying approaches.

4.1 Introduction

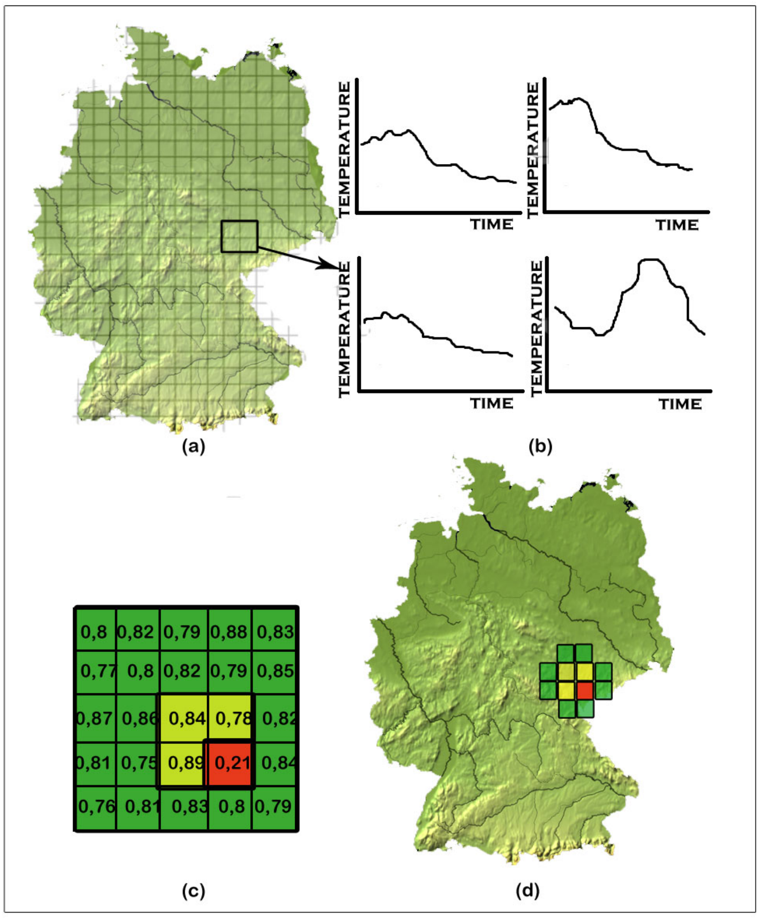


Figure 4.1: Example of our concept for mining temperature time series in the region of Germany. (a) shows the grid layout of MERRA for Germany, (b) measures the temperature of four specific, connected regions in this grid over time; (c) calculates the correlation coefficient between these temperature time series. The four selected grids are colorized in yellow or red. Red if the coefficient is an outlier, yellow else; (d) illustrates the matching of this found outlier (red) regarding its spatial region.

Both the current trends in technology such as smart phones, general mobile devices, stationary sensors and satellites as well as a new user mentality of utilizing this technology to voluntarily share information produce a huge flood of spatio-temporal data. Traditionally, a spatio-temporal database consists of triples (ID, time, location), mapping objects (e.g. users) and time to a position in geo-space where the object was, is, or will be located. In modern applications, sources of geo-spatial data, such like environmental sensor-data, each spatio-temporal point is further enriched with additional data. Consequently, mining this flood of data becomes an increasingly hard challenge. With huge environmental data like the open source Merra data set [167] from NASA we are suddenly able to make significant conclusions about the frequency of rare terms even in small spatial regions. By analyzing the spatio-temporal occurrences of these events, we can hope to find trends of regions when one region has a sudden significant outlier, that is not seen in the other regions in the vicinity. Such spatio-temporal outliers indicate some interesting events, like for example tropical storms.

In this work, we propose a framework to make this data actionable for classifying and predicting environmental events. An environmental event is a point in space and time enriched by a label describing the event, such as a storm or an earthquake. Given such an event in region s at time t, we propose to map this event into the Merra data set as a spatio-temporal environmental database to obtain information about the change of environmental attributes, i.e., their time-series, in spatial regions around s, and during the time around t. Explained by Tobler's first law of geography [184], we claim that the time-series of a region should be expected to be highly correlated to time-series of its vicinity, such that we can argue that any region that is not correlated to its vicinity indicate an outlier. Figure 4.1(a) illustrates our concept on the example of the Germany region. For each spatial grid cell we obtain a time series of measurements, such as temperature. As an example, the time-series of four individual regions are also depicted in Figure 4.1(b). As all four regions are located in very close vicinity, we assume that their time-series should be highly correlated. In this example, only three of the regions show a high spatial correlation value, as indicated by the correlation values shown in Figure 4.1(c). In a nutshell, each of these correlation values corresponds to the average Pearson correlation of the corresponding time-series to all of its spatial neighbors. We call these correlation values derived from their spatial neighbors an *Environmental Extremeness Measure* (EEM). If a EEM score is low, which contradicts Tobler's law, it indicates an outlier, as marked in Figure 4.1(d). We use this EEM to reduce a dense set of spatio-temporal environmental measurements to a sparse set of significant outliers. Given these sparse EEMs, we propose to use a tensor factorization, to reduce a large tensor, describing different environmental attributes over time and space, to a smaller set of latent features. Given labeled ground-truth events, we build a database of these tensors and their latent features, which we can use to build a model to classify these events.

To summarize, our contribution of this work are

- We propose an Environmental Extremeness Measure (EEM), which maps each point in space and time to a score value describing its local extremeness. Our outlier detection algorithm employs the EEM for deriving spatio-temporal outliers without prior knowledge of the data set.

- Using labeled event data, we propose two tensor factorization approaches to learn the latent factor to classify future events. We propose a simple 4-mode tensor approach, considering different labeled instances as a fourth mode, and we propose an approach using coupled tensor-tensor factorization.

- In our experiments, we apply our tensor factorization based learning approach, to a current problem in geo-information science: The problem of predicting rapid intensification of tropical cyclones. We show that our approach, by exploiting a spatio-temporal environmental database, is able to predict whether a tropical cylone will rapidly intensify its wind-speed by at least 30 knots in the next 24 hours. Our approach outperforms current literature on this problem.

We start with our unsupervised outlier detection approach in Section 4.2, where we define our EEM. The EEM is reused in Section 4.3, where we assume that historical events are labeled with ground-truth information that can be used for model training to similar events in the future. The experimental evaluation presented in Section 4.4 shows our results in predicting whether an tropical cyclone will increase at least 30 knots in 24 hours. Then we bring our work in the context of related work in Section 4.5 and conclude in Section 4.6.

4.2 Spatio-Temporal Outlier Detection

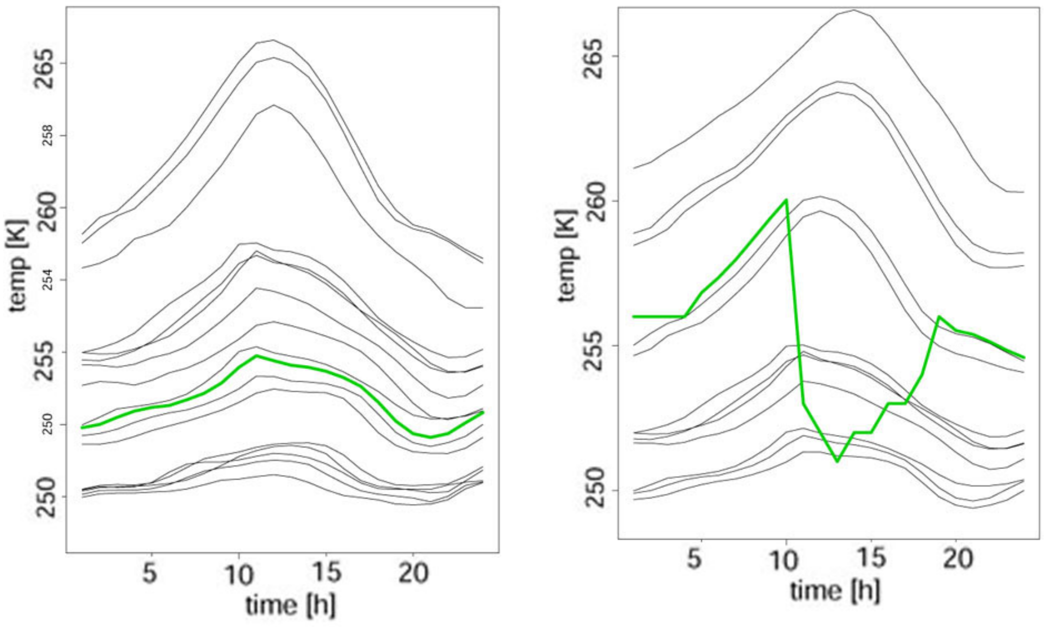


Figure 4.2: Example temperature data for 17 neighboring geo-locations. Every line corresponds to one location. The green line corresponds on the left hand to a closely correlated region compared to the others, on the right hand it is an extreme outlier with even negative correlation.

The aim of this section is to find regions whose attribute information, such as temperature differ significantly from other regions in the vicinity. This approach is used as a pre-processing step for our classification presented in Section 4.3, to identify and label anomalous environmental events in cases where these are not given by a authoritative ground-truth. To find outliers in a spatio-temporal database, we first need to provide a definition of a spatio-temporal database:

Definition 23 (Spatio-Temporal Database)**.** *Let $\mathcal{T}$ be a time domain, let $\mathcal{S}$ be a set of spatial regions, and let $\mathcal{A}$ be a set of attributes. A spatio-temporal database $\mathcal{D}$ is a collection of tuples (t, s, a), where $t \in \mathcal{T}$ is a point of time, $s \in \mathcal{S}$ is a spatial location, and $a \in A$ is an attribute value of domain $A \in \mathcal{A}$.*

Then we define a timeseries specifically for geographical regions:

Definition 24 (Attribute Time Series)**.** *Let $T \subseteq \mathcal{T}$ be a time interval, let $s \in \mathcal{S}$ be a spatial region, and let $A \in \mathcal{A}$ be an attribute domain, then $TS_{s,A,T}$ is a time series, i.e. a function mapping each point of time $t \in T$ to an attribute value $a \in A$ for a region $s \in S$. Thus, each region $s \in \mathcal{S}$ represents a specific time series:*

$$TS_{(s \in \mathcal{S}, A \in \mathcal{A}, T \in \mathcal{T})} : T \mapsto A$$

For detecting natural hazards finding outliers without prior knowledge of the given data set is the first crucial task. Our spatio-temporal outlier detection finds extreme regions using a calculated environmental extremeness measure (EEM). We define this measure by using Pearson correlation of a time-series to other time-series of regions in spatially close vicinity. In general consider the example in Figure 4.2, where the time series $TS_{s,A,T}$ of the attribute A =temperature is shown for 17 different regions s for some time interval T. Most of these curves show a similar behavior in terms of temperature change. Only the highlighted green time series shows little correlation to the other time series. As such it is regarded an outlier. To formalize such outliers, we define the notion of our EEM for spatio-temporal data as:

Definition 25 (Environmental Extremeness Measure)**.** *Let $T \subseteq \mathcal{T}$ be a time interval, let $A \in \mathcal{A}$ be an attribute domain, and let $s_1, s_2 \subseteq \mathcal{S}$ be two spatio-temporal regions, then*

$$Cor(TS_{(s_1,A,T)}, TS_{(s_2,A,T)}) :=$$

$$\frac{\sum_{t \in \mathcal{T}} (TS_{(s_1,A,T)}(t) - \overline{(TS_{(s_1,A,T)}(t))})(TS_{(s_2,A,T)}(t) - \overline{(TS_{(s_2,A,T)}(t))})}{\sqrt{\sum_t (TS_{(s_1,A,T)}(t) - \overline{(TS_{(s_1,A,T)}(t))})^2 \sum_t (TS_{(s_2,A,T)}(t) - \overline{(TS_{(s_2,A,T)}(t))})^2}}$$

denotes the correlation of the time-series of regions s_1 and s_2 in attribute A during time T, where $\overline{(TS_{(s_1,A,T)}(t))}$ denotes the mean *of all values, respective for region s_1. For a set of spatial regions $S \subseteq \mathcal{S}$, we define our environmental extremeness score of a region s as the average correlation to all time-series of S, formally:*

$$EEM(s, S, A, T) = \sum_{s' \in S} Cor(TS_{(s,A,T)}, TS_{(s',A,T)})/|S|$$

The current literature defines outlier and its difference from other points in terms of distance and density [62, 117]. For spatio-temporal objects, we need to adapt this definition. Clearly, we could employ a time-series distance to measure the similarity of two spatio-temporal regions (s_1, t_1) and s_2, t_2, such as Euclidean distance or Dynamic Time Warping [25]. However, this might not be possible as two regions may have similar trends of contextual information over time, but their absolute values may be different as illustrated in the following example.

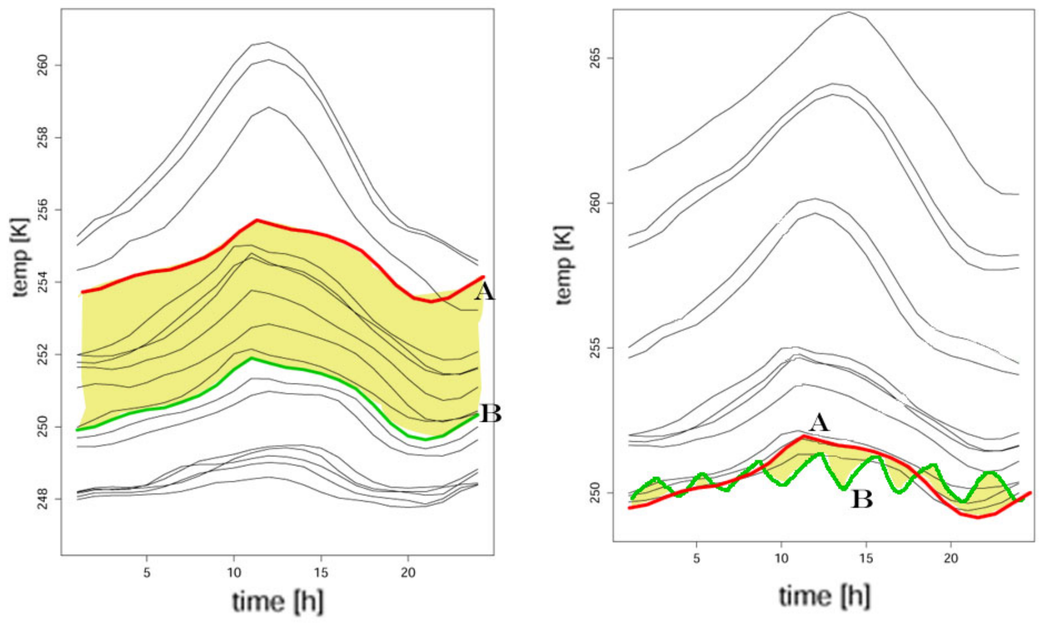


Figure 4.3: Correlation VS. DTW and Edit Distance. Correlating the red and green temperature curves would remark (a) as most similar curves. Using DTW or Edit Distance would resolve (b) as the most similar as the covered area (yellow) is smaller.

Example 8. *Given in Figure 4.3 are two time series A, B showing the temperature (in Kelvin) to different time frames. A and B in Figure 4.3(a) are showing the same functional behavior for different regions as A is for example measured temperature in a valley while B is showing the temperature on a mountain. Thus, their actual base temperature do not share any similarity. However in Figure 4.3(b), time series B starts at around the same temperature level as time series A but does show a quite different altitude over time. Now, Dynamic Time Warping and Edit distance would both chose the two time series in Figure 4.3(b) as the most similar time series as their temperature values differ less between and the space between A and B (yellow) is minimized. But regarding Definition 25 we would want our measure* *to regard the temperature curves in Figure 4.3(a) as similar (even as exact match) because the amount the temperature* changes *between these two time series are the same. Correlation reasonably asseses the similarity of A, B, and C matching our intuition and the the application-specific requirements from geospatial data. Moreover, correlation is efficient to compute.*

With this stated we define an outlier and the corresponding spatio-temporal outlier the following:

Definition 26 (Outlier)**.** *An outlier is a point, which varies sufficiently from other points such that it appears to be generated by a different process from the one governing the other points.*

Following Tobler's first law of geography, two regions are expected to have a similar context (attribute values) if they are spatially close. This allows a straightforward extension of the general concept for outliers for spatio-temporal outlier:

Intuitively, a region $s \subseteq S$ is called a spatio-temporal outlier, if the attribute A in region s during time interval T, is so different from the attribute at time T in regions spatially close to s, such that the suspicion arises, that it appears to be generated by a different process. More formally, we define a spatio-temporal outlier as follows.

Definition 27 (Spatio-Temporal Outlier)**.** *Let $\mathcal{D}$ be a spatio-temporal database, let τ be a real value, let $s \in \mathcal{S}$ be a spatial region and let $S \subseteq \mathcal{S}$ be a set of spatial region close to s. Region s is called a spatio-temporal outlier for attribute A during time T if*

$$EEM(s, S, A, T) \leq \tau.$$

According to Definition 27, a spatio-temporal outlier is a region whose average EEM value is below a user specified threshold τ. The automated choice of parameter τ, which should be chosen individually for each attribute domain $A \in \mathcal{A}$, is beyond the scope of this chapter. In our experimental evaluation, we chose this parameter empirically.

4.2.1 Algorithmic procedure

Our overall goal is to find spatio-temporal outlier for a specific region. By applying Toblers Law of Geography [184]. We find this region by comparing the correlation

Algorithm 1. Average Correlation.

```
Input: region s, time interval T, attribute A
1: totalCor:=0
2: for $s'$ in Radius(s,r) do
3:     totalCor+=Cor($TS_{(s,A,T)}, TS_{(s',A,T)}$)
4: end for
5: return $\frac{totalCor}{Radius(s,r}$.
```

Algorithm 2. Spatio-Temporal Outlier Detection.

```
Input: time interval T, attribute A, double $\tau$
1: results ← $\emptyset$
2: for s in $\mathcal{S}$ do
3:     for t in $T$ do
4:         for A in $\mathcal{A}$ do
5:             if AverageCorrelation(s,T,A) $< \tau$ then
6:                 result ← result $\cup s$
7:             end if
8:         end for
9:     end for
10: end for
11: return results.
```

between this one region to the others surrounding regions by using the formula in Definition 25 on each of them. This correlation calculated with knowledge of the vicinity of the region is called an environmental extremeness measure (EEM). To restrict the areas of these surrounding regions we have to define a radius in which we regard the regions as connected:

Definition 28 (Spatial Radius)**.** *Let $r \in \mathbb{N}$. Let s be a region, then $Radius(s, r)$ is the set of spatial regions having a distance smaller than r.*

With this definition we are able to compute the correlation as shown in Algorithm 1. Algorithm 1 computes, for a given time interval T, a given Attribute A, and a given spatial region s, the average correlation between s and the set of its spatial neighbors Radius(s,r).

After that we give the procedure for the overall spatio-temporal outlier detection in Algorithm 2. For a given time interval T and an attribute A, and an outlier threshold τ, Algorithm 2 computes the set of outlier regions in attribute A during time T.

4.3 Framework for Natural Hazards Detection

Our supervised classification approach assumes labeled ground-truth hazard information, such as the time and location of historical topical cyclones. These can be obtained either from a natural hazard database such as the NOAA natural hazard database [1] or by employing the unsupervised outlier detection approach presented in Section 4.2. Thus, in the following, we assume a set GT of labeled ground-truth events, such that each element $(s, A, T) \in GT$ describes a hazard at a spatial location s, during a time interval T, using a set of attributes A. In the following we will discuss the second main building block of our framework: the tensor factorization and classification.

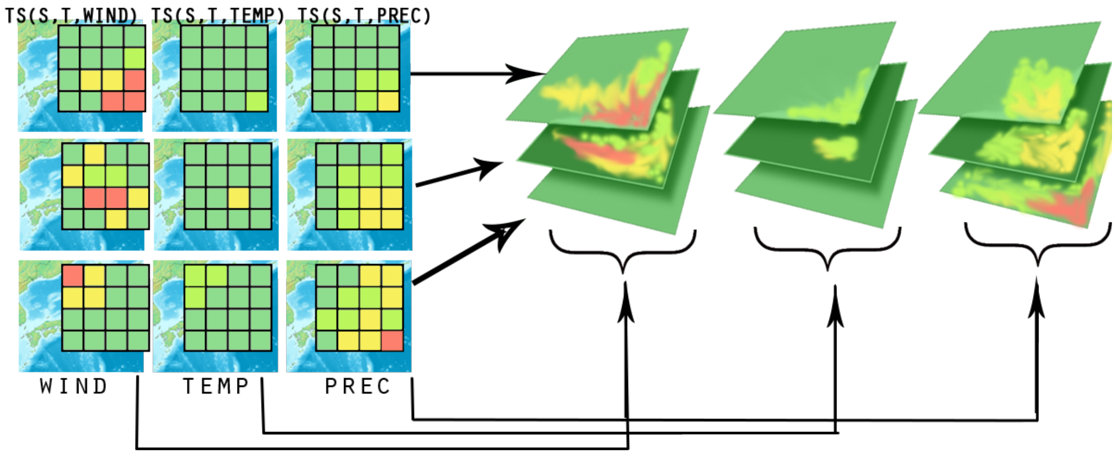


Figure 4.4: Creating a tensor from precipitation, temperature and wind speed data.

4.3.1 Environmental Tensor Factorization

For our supervised framework the exact position (latitude, longitude) of possible labeled outliers and their vicinity are taken to form an environmental tensor X, formally:

Definition 29 (Environmental Tensor)**.** *Let* $(s, \mathcal{A}, T) \in GT$ *be an event, and let* $S = \{s_1, ..., s_n\}$ *be a set of spatial regions around the event region* s. *Then we*

[1] https://www.ngdc.noaa.gov/hazard/

use the set of time series $TS(s \in S, T, A \in \mathcal{A})$ to construct a space-time-attribute tensor as follows:

$$X(1 \leq i \leq |S|, 1 \leq j \leq |\mathcal{A}|, 1 \leq k \leq |T|) = TS(s_i, A_j, T)(k)$$

Intuitively, $X(i,j,k)$ is the k'th value of the time series of attribute A_j in region s_i. For a specific point in region s_1, a set of spatial regions $S = \{s_1, ..., s_n\}$ in spatial vicinity of s, a time interval T, and attribute domains $\mathcal{A} = \{A_1, ..., A_{|\mathcal{A}|}\}$, the set of time series $TS_{(s \in S, A \in \mathcal{A}, T)}$ constitutes the three dimensional tensor X.

Figure 4.4 depicts the construction of *one* environmental tensor. For each of the three considered attributes $A \in \mathcal{A}$ and for each of the 16 considered spatial regions $s \in S$, we obtain a time series. A 3-mode tensor X is created from these time series by, concatenating the time series of each spatial location, yielding a two dimensional space-time matrix, which are concatenated over all attributes, yielding a space-time-attribute tensor corresponding to the tensor defined in Definition 29.

To derive latent features from such an environmental tensor, we employ a tensor factorization.

Definition 30 (3-Mode Tensor Factorization)**.** *A classical CANDECOMP/-PARAFAC [105][158] tensor factorization decomposes a tensor $X(S, T, \mathcal{A})$ as follows*

$$X(S, T, \mathcal{A}) = U \circ V \circ W + error$$

where $\circ$ denotes the 3-mode tensor product ([119]), U is a $|S| \times K$ matrix, V is a $|T| \times K$ matrix and W is a $|\mathcal{A}| \times K$ matrix. The parameter K is a parameter of the tensor factorization, controlling the number of latent features extracted for each mode. Furthermore, $error$ corresponds to the loss of information incurred by factorizing a large three mode tensor into three small matrices.

The 3-mode tensor defined above represents the environmental information of a single labeled region. For creating a model of all existing labeled regions and their vicinity we want to decompose *all* environmental tensors simultanously. This can be done in two ways: First, by making use of the special case that all environmental tensors have all three modes in common by applying a Four-Mode Tensor Factorization:

Definition 31 (Environmental Four-Mode Tensor)**.** *For n environmental tensors $X_1(s_1, t_1, \mathcal{A}_1), ..., X_n(s_n, t_n, \mathcal{A}_n)$, the environmental four-mode tensor $\mathcal{X}$ is defined as:*

$$\mathcal{X}(1 \leq i \leq |S|, 1 \leq j \leq |\mathcal{A}|, 1 \leq k \leq |T|, 1 \leq l \leq n) = X_l(i, j, k)$$

having

$$\mathcal{X}(U, V, W, ID) = U \circ V \circ W \circ ID + error$$

as the four-mode tensor factorization (4MTF) with U, V, W defined as in Definition 30, $\circ$ denoting the 4*-mode tensor product, and ID is a $n \times K$ matrix describing latent features of each environmental tensor, and thus, of each environmental 3-mode tensor stored in the 4-mode tensor $\mathcal{X}$.*

Besides the 4MTF where all three modes of the environmental tensors are coupled, it is possible to couple only one dimension of these tensors. This is done by coupled tensor-tensor factorization (CTTF), which works similar for this case as the coupled matrix-tensor factorization (CMTF) [2].

Definition 32 (Coupled Tensor-Tensor Factorization)**.** *Given n environmental tensors $X(s, T, \mathcal{A})$ and let them be sharing only the attribute mode $\mathcal{A}$, their decomposition optimizes:*

$$\frac{1}{2}\|s - (A_1 \cdot A_2... \cdot A_n)\|^2 \|T - (A_1 \cdot A_2...A_n)\|^2$$

where s and T are the variable modes of the environmental tensors and $A_1...A_n$ is the number of fixed matrices over all environmental tensors for each attribute domain A. More details on coupled factorization can be found in [2].

For the collective factorization of matrices, coupling is crucial in at least one mode. Else the latent factors (concepts) of the environmental tensors would be independent and thus incomparable. Semantically it is different to lock either only one mode (CTTF) or three modes as we will evaluate in the experimental section 4.4. The overall goal of the environmental tensor factorization is:

- finding discriminative latent attributes describing the environment around labeled environmental hazards,
- reducing the dimensionality of the environmental tensors,

- finding similarities among different spatial regions using matrix U, among different time intervals using matrix V, among different environmental attributes using matrix W, and among different environmental events using matrix ID.

In the next subsection, we employ the two tensor factorization approaches 4MTF (Definition 31) and CTTF (Definition 32) to classify new unlabeled environmental hazards given a set of labeled environmental hazards, using the environmental tensors extracted using knowledge about time and location of the labeled hazards.

4.3.2 Classifying Natural Hazards

After the environmental tensor factorization, we employ a black box of classifiers to model historic natural hazards that can predict future occurrences of such hazards basing on the historic data. We note that these classification algorithms are not the main focus of this work. Depending on which tensor factorization is used we have two different inputs for a given classifier:

- CTTF: all uncoupled modes, in this case the space s and time T.
- 4MTF: in the 4-mode case, the modes U, V, W describe the *common* aspects between *different* tensors regarding their attributes/time/space. The ID, in contrast, describes for *each* tensor its *individual* aspects. thus, using these individual aspects might lead to a discrimination between the tensors and can thus be used as input features to a classifier.only the fourth mode ID is uncoupled and as such used as input for the classification.

These latent features can be fed to traditional classification algorithms. The quality of these algorithms on our factorized latent features will be explored in the next section.

4.4 Experimental Evaluation

Our experimental evaluation includes two steps: (a) an evaluation of our unsupervised spatio-temporal outlier detection method using data where we synthetically added outliers to a sample set of real MERRA data and (b) an evaluation of our full framework applying unsupervised approaches like the outlier detection from (a) as well as supervised approaches like tensor factorization and classification. The full framework is not only capable of finding outliers but works also as a multi-solution tool for attribute selection and mining massive data. We start by introducing the used large real data set.

4.4.1 Global Climate Data

The open source data collection MERRA, used in this chapter, is provided by NASA [166] and consists of more than six gigabytes of spatio-temporal environmental data per day, having collected more than 70 Terra bytes of data in the last years.[2] The MERRA time period covers the modern era of remotely sensed data, from 1979 through the present, and covers a large variety of hundreds of environmental parameters on a spatial resolution of 0.5 degrees latitude times 0.67 degrees longitude produced at one-hour intervals.

For our supervised classification experiments, we used the *Statistical Hurricane Intensity Prediction Scheme (SHIPS)* database [56, 200]. This database contains location and time of 800 tropical cyclones (TCs) from 1984-2011. Each TC is recorded every six hours. Each observation of a TC is also enriched with many attributes of the TC, such as wind-speed and air-pressure. The NCAR/NCEP reanalysis data is the one most trusted one among geographers in the last years and covers 40 years of environmental data generated by a frozen state-of-the-art global assimilation system and a database as complete as possible. They use a static forecast/analysis system and perform data assimilation using past data for the time of 1957 to 1996. [3]. Last, the Earth Science Data consists of daily global air temperature and precipitation measurements, aggregated from heterogeneous sensors for fifty years (1950-1999). With less attributes than the other data sets this data set still huge due to fine grid granularity. Each sensor corresponds to a physical location (latitude, longitude) on the Earths surface. The data is downloadable in csv format from the Oak Ridge National Laboratory website.

Thus, the initial challenges of knowledge discovery, such as data selection, data preprocessing, data integration and data transformation have been solved, it is time for the data mining community to step in to advance the research frontier by finding useful and previously unknown patterns in earths environment.

[2]`https://gmao.gsfc.NASA.gov/products/documents/MERRA_File_Specification.pdf`
[3]`http://rda.ucar.edu/datasets/ds090.0/docs/publications/bams1996mar/bams1996mar.pdf`

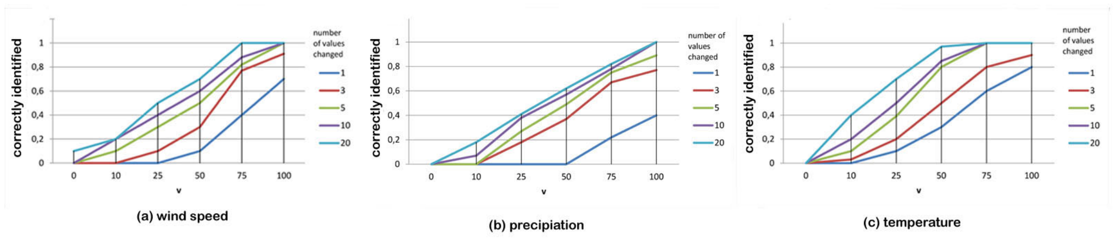


Figure 4.6: Synthetic outlier data varying the number of manually changed values for the attributes temperature, precipitation and wind speed.

4.4.2 Finding synthetic spatio-temporal outliers

The first step for proving our concept for spatio-temporal outlier detection is showing whether we are generally able to find given outliers. Since outliers, i.e., triples of space s, time t and attribute A, cannot be verified easily without traveling back in time to t, we generate synthetic outliers. For this purpose, we randomly *distort* real time series, by changing the corresponding attribute values. We distort a time-series in two ways: (i) the number of values that are changed in the time series, ranking from a single changed value up to 20 changed values; and (ii) the magnitude v of the change itself, i.e. how strong is the original value altered, ranging from 10 to 100 of the original value. The effect of this change is of course also regarded to the given measurement scale of the three attributes. Temperature is in Kelvin, wind speed in km/h and precipitation in mm/h. The radius for the size of the vicinity is set to a manhattan distance of 1.

Figure 4.6 shows the result of our outlier detection for three attributes *Wind Speed*, *Temperature* and *Precipitation*. We measure the fraction of distorted time-location pairs for which we distorted the corresponding time series. As expected, we observe that a larger magnitude v of distortion increases the fraction of distortions found as outliers. At the same time, an increased duration of distortion, specified by the number of values changed, also improves the detection rate. Overall, we can see that our unsupervised outlier detection approach is able to detect environmental time series, whose attribute values are sufficiently distorted in terms of the number of time series values changed and in terms of magnitude of this change, with high accuracy. However, the more applicative problem of supervised classification of environmental events, given a set of labeled events, is evaluated in the next section.

Synthesizing Wind Speed

Depicted in Figure 4.6(a), ouf of 100 tries outliers 0.74 percent of outliers can be found which is close to the maximum 0.8 percent achieved by temperature and thus a very good result as the temperature data set is much less versatile than wind speed. All outliers are found by changing the number of values v to 75 for 20 different points in time.

Synthesizing Temperature

Due to the low variability, the temperature attribute shows the best results for finding synthetic outliers in the data set. Figure 4.6(b) shows clearly, that the lowest values achieved by just altering a single value reaches up to 0.8 percentage of outliers are recognized with the highest increased value, while the highest number of changed values in the time series already achieves nearly maximal found outliers $0,98$ percent for only added half the value (50).

Synthesizing Precipitation

Although the most versatile data set, precipitation gains the least results compared to the other two attributes. As shown in Figure 4.6(c), precipitation reaches for single values only a maximum of 0.38 percent of correctly identified outliers and all outliers are only correctly identified for the highest changed value v and the highest amount of time points.

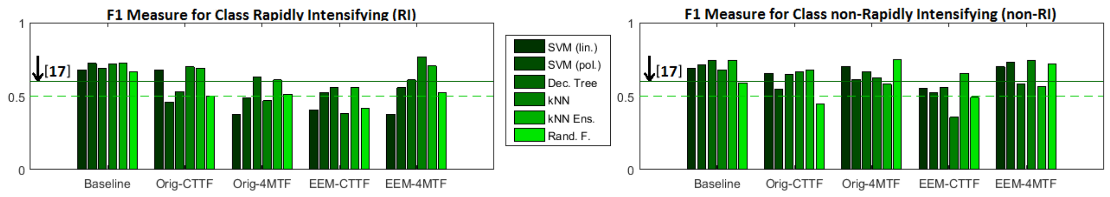


Figure 4.7: Tropical Cyclone classification results.

4.4.3 Finding Natural Hazards on Real Data

Our experimental evaluation for the supervised approach aims at classifying Rapid Intensification (RI) of tropical cyclones (TC) [111] in the Atlantic ocean using MERRA data. For each TC in the SHIPS dataset, at time t and location s, we obtained an environmental tensor (as defined in Definition 29) from the MERRA dataset as follows: We queried the MERRA dataset for the 24-hour time interval before t, in all 25 spatial regions having a manhattan distance of at most two to s, using all 387 environmental attributes available in MERRA. This way, we were able to link each TC available in the SHIPS data set, to a size $25 \times 24 \times 387$ tensor of environmental measurements. We labeled each TC as “RI” if the wind speed of the TC increased by at least 30 knots within the next 24 hours, or “non-RI” otherwise. About 5% of TCs were labeled as “RI” this way. The task is to predict this label. We evaluated the quality of our proposed algorithms using F1-measure on the two classes “RI” and “non-RI”. For reference, according to [112], the rapid intensification forecasting from the National Hurricance Center (NHC) official forecast at 24 hour lead time for the Atlantic Basin reached only a 10% probability of detection with more than a 30% false alarm rate. We evaluate the following algorithms.

- Competitor [200]: A data mining approach recently presented in [200], which uses only data in the SHIPS data set, which includes less than two hundred parameter per TC, and uses standard classification algorithms on these features. According to the results of [200], this approach had a F1-measure of no more than 60% for both classes in any experiment.

- Baseline: The baseline is provided by putting the raw data, without any dimensionality reduction (4MTF or CTTF) into the classifiers.

- Orig-CTTF and Orig-4MTF: These two algorithms apply our two tensor factorization approaches (either CTTF or 4MTF) on the raw data tensors, and use the resulting latent features for classification.

- EEM-CTTF and EEM-4MTF: These two algorithms use the full framework, thus applying the tensor sparsification using our environmental extremeness measure (EEM). Then, tensor factorization (CTTF or 4MTF) is applied to this pre-processed tensor and the resulting latent features are used for classification.

For the classification step, we use the following standard Matlab implementations, using 10-fold cross validation. If the parameter setting is not provided, it is kept at the standard given by Matlab:

- SVM support vector machine with linear kernel.

- SVM support vector machine with polynomial kernel.
- Decision Tree. Decision Tree classification.
- k-nearest neighbors (*knn*) classification using $k = 10$ nearest neighbors.
- kNN Ensemble using random subspaces and $k = 10$ in each subspace.
- Random Forrest Ensemble using RUSBoost.

In all cases except EEM-CTTF and EEM-4MTF, the data was standardized before using the classifiers. For these two experiments this was not necessary as the EEM already normalizes the input for the classifiers. In addition, the algorithms were set up to take the prior distribution of the labels into account.

Results

Our tropical cyclone rapid intensification classification results are shown in Figure 4.7, depicting the F1 measure for the two classes "RI" and "non-RI". Overall, we can see that all proposed approaches using the full information contained in the spatio-temporal tensor of a TC, including the baseline, yield a higher classification quality than the classification of features of the SHIPS dataset in [200], which showed a F1-measure of no more than 60% in any experiment. This promising result proves that our concept, of joining a natural hazard database (such as the SHIPS database) with a spatio-temporal environmental database (such as MERRA), has the potential to improve the classification and forecast of natural hazards beyond the state-of-the-art of domain-specific solutions, by exploiting the wealth of public environmental data.

Comparing the five approaches using spatio-temporal tensors obtained from MERRA, we note that our baseline solution performs very well. In fact, our solutions using tensor factorization are not able to outperform our baseline classification for many classification algorithms in many settings. This indicates that our four tensor factorization based algorithms still allow room for future research and improvement. Still, see that our most sophisticated approach *EEM-CTTF*, which employs our environmental extremeness measure (cf. Section 4.2) and the coupled Tensor-Tensor factorization (c.f. Section 4.3) is able to achieve a much higher classification rate using Random Forests, than any other algorithm can achieve using the baseline features. This improvement is quite significant, and shows that our approach of factorizing space, time and environmental attributes can indeed be used to learn and classify complex environmental phenomena, such as the prediction of whether a TC will rapidly intensify in wind-speed.

4.5 Related Work

Large spatio-temporal environmental data has created an enormous interest in findings patterns in earth climatic changes. In [72], the authors introduce climate challenges to the data mining community and compare and contrast climate data mining to spatial and spatio-temporal data mining.

Steinbach *et al.* [178] argues to apply principal components analysis (PCA) and singular value decomposition (SVD), to discover climate indices. Our outlier detection algorithm is closest related to [53], where a neighborhood based outlier detection algorithm is provided. Unlike us, they detect outliers over space and time only *separately* to explain some of the extreme events like drought and severe rainfall at specific locations on earth. For the use case of finding tropical cyclones, [201], Yang *et al.* used association rule mining to look for combinations of persistent and synoptic conditions which provide improved RI (Rapidly Intensifying) probability estimates. Similar in [202], the authors apply association rule mining on the analysis of intensity changes of North Atlantic tropical cyclones. As mentioned earlier, in addition to outlier detection, classification algorithms can be useful when trying to find, characterize and especially predict natural hazards. For instance, [171] provides examples for classification being used in combination with spatio-temporal data. More specifically, these works spatio-temporal data obtained from satellite images to classify types of vegetation. Clearly, identifying vegetation using our framework of using environmental tensor will be an interesting new case study. In [55], data from the weather forecast model Eta was used to identify patterns which can be associated with unusual weather activity. Wang and Ding [190] use classification in their three step approach to build a forecasting model for extreme rainfalls using using location-based patterns. This approach can be used alternatively to our EEM-measure to find occurrences of spatio-temporal outliers on an unlabeled dataset. A comparative study, using different environmental outlier detection approaches is planned for our future work. The survey in [119] gives an overview over implementations of tensor factorizations used in this work.

4.6 Conclusion

Concluding, we presented a framework for detection and prediction of environmental hazards. To predict future hazards, given labeled historic ground-truth of historic hazards, our framework allows to learn latent features in time, space, and attributes using tensor factorization. Our framework allows to predict any kind of hazard, including complex phenomena such as rapid intensification of tropical cyclones. Our experimental evaluation shows that our framework has the potential to revolutionize the state-of-the-art in this field. This result is not contributed to our data mining algorithms, which are all state-of-the-art. Rather, this result is the consequence of joining an existing problem, with the vast amount of spatio-temporal environmental data publicly available in the MERRA database.

CHAPTER 5

Data Mining in Main Memory DB Systems

Parts of this chapter have previously been published in LWA 2015 [183] and EDBT 2017 [159].

The last chapter of this thesis spans a topic that covers all the methods introduced in the previous chapters. How can all these methods be efficiently integrated into database systems if you want to apply these methods inside the DBMS? What achievements are given by doing that or is it even sensible? We like to answer these questions and even more with the following research on the example of the database HyPer and three classical data mining / machine learning approaches: classification (Naive Bayes [126]), clustering (k-Means [131]) and graph traversals (PageRank [34]).

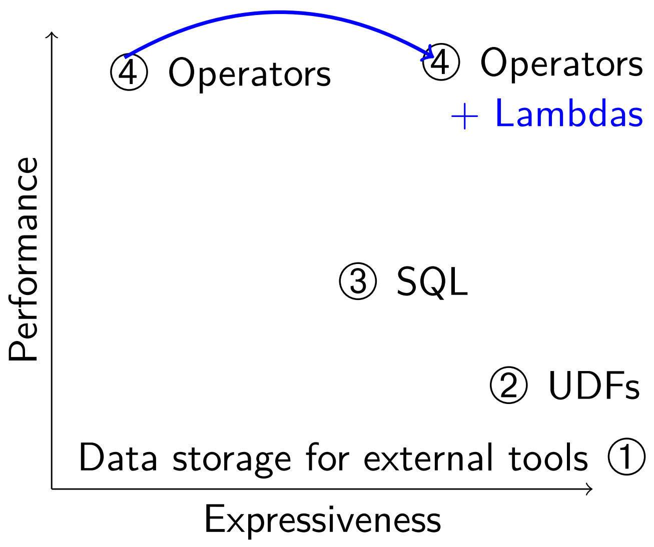


Figure 5.1: Overview of approaches to data analytics using relational database systems. Our system supports the novel *layer 4*, where data mining is integrated directly into the database core, thus, leading to higher performance. To maintain expressiveness, high-order functions (*lambdas*) can directly be passed as parameters to the database operators.

5.1 Introduction

The current data explosion happening in science and technology poses difficulties for data management and data analytics. Especially stand-alone data analytics applications [133, 86] are prone to have problems due to their simple data management layer: Being optimized for read-mostly or read-only analytics tasks, most stand-alone systems are unsuitable for frequently changing datasets: After each change, the whole data of interest needs to be copied to the application again, which is a time and resource consuming process.

We define *data analytics* to be algorithms and queries that process the whole dataset (or extensive subsets), and therefore are computation-intensive and long-running. This domain contains for example machine learning, data mining, graph analytics, and text mining. Beside the differences between these subdomains, most algorithms boil down to a model-application approach, i.e., a two phase process where a model is created and stored first, and then applied to the same or different data in a second step.

In contrast to the afore-mentioned dedicated analytical systems, classical RDBMSs provide an efficient and update-friendly data management layer and many more useful features to store big data reliably, such as user rights management and recovery procedures. Besides, database systems avoid data silos, as data has to be stored only once, eliminating ETL cycles (extraction, transformation, and loading of data). Thus, we investigate how data analytics can be sensible integrated into RDBMSs to contribute to a "one-solution-fits-it-all" system. What efficiency is possible when running such complex queries in a database? Can it actually be better than single purpose standalone systems? According to Aggarwal et al. [9], seamless integration of data analytics technology into DBMSs is one of the most important challenges.

Some newer database systems, for example SAP HANA [66] and HyPer [115], are already designed to efficiently handle different workloads (OLTP and OLAP) in a single system. Main-memory RDBMSs like HyPer are specifically well-suited for high analytics workload due to their efficient use of modern hardware, i.e., multi-core CPUs with extensive instruction sets and large amounts of main memory.

Still, another questions remains: How is an analytics algorithm best integrated into an RDBMS? While existing database systems that feature data analytics include the algorithms on a very high level, we propose to add a specific set of algorithmic building blocks as deep in the system as possible. To describe and assess different approaches of integrating data analytics algorithms into an RDBMS, we distinguish four layers, where the fourth layer is the most deeply integrated:

(1) DBMS as data storage with external analytics algorithms—the nowadays most commonly used, but least integrated approach.

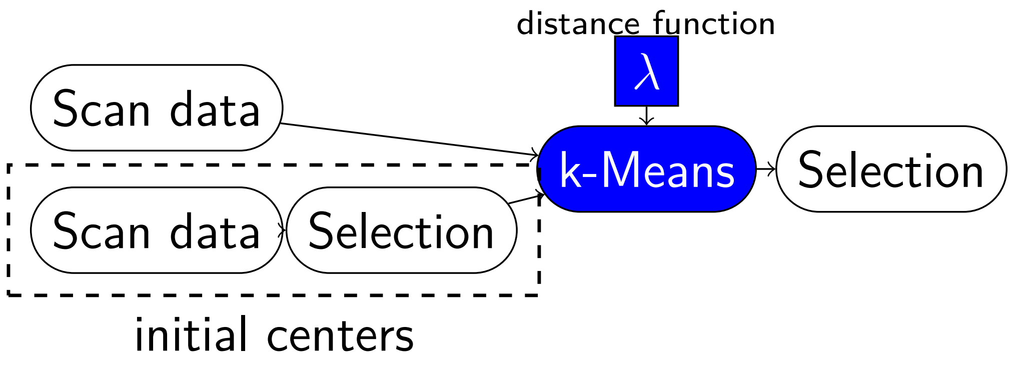
(2) User-defined functions (UDFs)—code snippets in high-level languages executed by the DBMS.

(3) SQL queries—including recursive common table expressions (CTE) and our novel iteration construct.

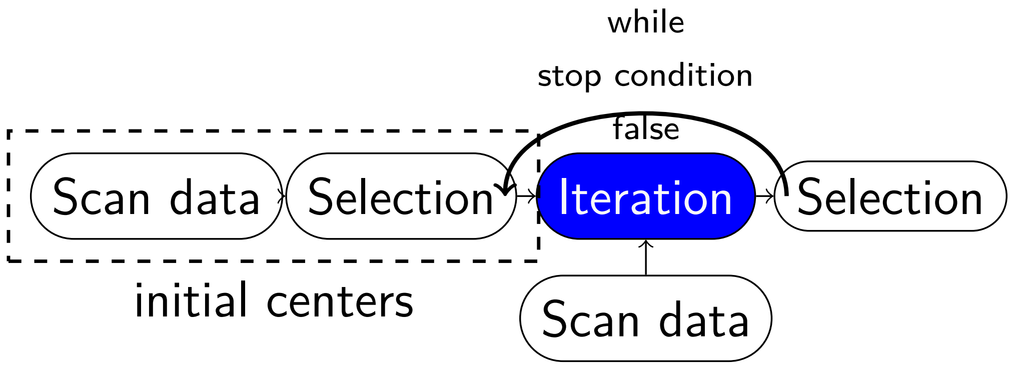
(4) Integration as physical operators—the deepest integration, providing the highest performance.

To increase flexibility within *(4)*, we propose user-defined code snippets as parameters to our operators. These so-called lambda functions, containing for instance distance metrics, are able to change the semantics of a given analytical algorithm.

All these approaches trade performance versus flexibility in a different way, as depicted in 5.1. We propose implementing multiple of those approaches into one system to cover the diverse needs regarding performance and expressiveness of different user groups and application domains. The novel operator integration *(4)* combines the highest performance with high flexibility, but can only be implemented by the database system's architects, while *(2)* and *(3)* provide environments in which expert users can implement their own algorithms. Furthermore, all three integrated approaches *(2)-(4)* avoid ETL costs, stale data, as well as assembling and administrating complex system environments, and therefore facilitates ad-hoc data analytics.



(a) Operator-centric approach. The iterative k-Means algorithm is implemented as physical relational operator. The distance function is specified as a lambda expression.



(b) SQL-centric approach. The iterative algorithm, including initialization and stop condition, is expressed in SQL. The iteration operator can either be the standard recursive common table expression, or our optimized non-appending iteration construct.

Figure 5.2: Simplified query plans for k-Means clustering using the operator-centric and SQL-centric approach to data analytics in databases.

5.1.1 Contributions

In this chapter, we present how data analytics can efficiently be integrated into relational database systems. Our approach targets different user groups and application domains by providing multiple interfaces for defining and using analytical algorithms. Our contributions are the following:[1]

- A classification and assessment of approaches to integrate data analytics with databases.
- The iteration construct as extension to recursive common table expressions (`with recursive`) in SQL .
- Analytical operators executed within the database engine that can be parameterized using lambda expressions (anonymous user-defined functions) in SQL.
- An experimental evaluation with both dedicated analytical systems and database extensions for analytical tasks.

5.1.2 Structure

The remainder of this chapter is organized as follows: An overview of the related work is given in 5.2. After that, in 5.3 we discuss what characteristics make HyPer especially suited for in-database analytics. We continue by explaining the in-database processing in 5.4. How analytics can be integrated into SQL is defined in 5.5 and our building blocks (operators) of the fourth layer are shown in detail in 5.6. Our operators are very flexible due to their lambda functions, which we describe in 5.7. The evaluation of our operators in HyPer is given in 5.8. Last we conclude our work in 5.9.

5.2 Related Work

Data analytics software can be categorized into dedicated tools and extensions to DBMSs. In this section, we introduce major representatives of both classes.

[1]Partially based on our prior publication [183].

5.2.1 Dedicated Data Analytics Tools

The programming languages and environments *R*[2], *SciPy*[3], *theano*[4] and *MATLAB*[5] are known by many data scientists and are readily available, hence, they are heavily used for data analytics, and implementations of new algorithms are often integrated into their libraries by researchers. In addition, these languages and environments provide data visualizations and are, thus, well-suited for exploration and interactive analytics. However, their algorithm implementations often are only single-threaded, which is a major drawback concerning today's multi-core systems and data volumes.

The next group of existing data analytics tools are *data analytics frameworks*. Most representatives of this category are targeted at teaching and research, hence, they do not focus on performance for large datasets. Their architecture makes it easy to implement new algorithms, and to compare different variants of algorithms regarding quality of results. Notable examples include *ELKI*[6], supporting diverse index structures to speed up analytics, *RapidMiner*[7], used in industry as well as research and teaching, and *KNIME*[8], where users can define data flows and reports via a GUI.

Recently, Crotty et al. presented *Tupleware* [48], a high-performance analytical framework. Tupleware is meant for purely analytical tasks, hence, the system is not taking into account transactional data. The authors endorse interactive data exploration, e.g., by not relying on extensive data preparation [49], and by providing a data exploration GUI. Tupleware requires users to annotate their queries with as much semantics as possible: Queries may solely consist of simple building blocks, e.g., `loop` or `filter`, augmented with user-defined code snippets such as comparison functions. Relational operators—the building blocks of SQL queries—are fairly similar to Tupleware's building blocks, but are more coarse-grained, more robust against faulty or malicious user input, and can be used in a more general fashion. They therefore do not guarantee as many invariants. SQL implementations of algorithms, hence, could be optimized in a similar fashion, although this requires major changes to relational query optimizers.

[2]`http://www.r-project.org/`
[3]`http://www.scipy.org/`
[4]`http://deeplearning.net/software/theano/`
[5]`http://www.mathworks.com/products/matlab/`
[6]`http://elki.dbs.ifi.lmu.de/`
[7]`http://rapidminer.com/`
[8]`http://www.knime.org/`

The cluster computing framework *Apache Spark* [205] supports a variety of data analytics algorithms. Analytical algorithms, contained in the Machine Learning Library (MLlib), benefit from Spark's scale-up and scale-out capabilities. *Oracle PGX*[9] is a graph analytics framework. It can run predefined as well as custom algorithms written in the *Green-Marl* DSL, and is focused on a fast, parallel, and in-memory execution. *GraphLab* [132] is a machine learning framework that provides many machine learning building blocks such as regression or clustering, which facilitate building complex applications on top of them. As all these frameworks use dedicated internal data formats that make necessary time-consuming data loading steps. Furthermore, the synchronization of results back to the original RDBMS is a complex job that often must be implemented explicitly by the user.

5.2.2 Data Analytics in Databases

Besides standalone systems there are database systems which contain data analytics extensions. Being faced with the issue of integrating data analytics and relational concepts, the systems mentioned below come up with different solutions: Either analytical algorithms are executed via calls to library functions, or the SQL language is extended with data analytics functions.

MADlib [91] is an example for the second level our our classification, user-defined functions. This library works on top of selected databases, and makes heavy use of data parallel query execution, if provided by the underlying database system. MADlib provides analytical algorithms as user-defined functions written in C++ and Python that are called from SQL queries. The underlying database executes those functions, but cannot inspect or optimize them. While the output produced by the functions can directly be post-processed using SQL, only base relations are allowed as input to data analytics algorithms. Thus, full integration of the user-defined functions and SQL queries is neither achieved on a query optimization and execution level nor in the language and query layer.

Another example for the UDF category is the *SAP HANA Predictive Analytics Library (PAL)* [172, 66]. This library offers multi-threaded C++ implementations of a variety of analytical algorithms to run within the main-memory database system SAP HANA. It is integrated with the relational model in a sense that input parameters, input data, as well as intermediate results and the output are

[9]`http://www.oracle.com/technetwork/oracle-labs/parallel-graph-analytics/`

relational tables. The algorithms, so-called *application functions*, are called from SQL code. They are compiled into query plans and executed individually. In contrast to the afore-mentioned MADlib, it integrates in one ecosystem only, and is therefore capable of connecting to SAP HANA's user and rights management.

Oracle Data Miner [181] has its focus on *supervised* machine learning algorithms. Hence, training data is used to create a model before this model is applied to test data using SQL functions. Both steps are run multi-threaded to make use of modern multi-core systems. Results of the algorithms are stored in relational tables. Interactive re-using and further processing of results within the same SQL query is not possible, but can be applied in precedent and subsequent queries.

LogicBlox [17] relies on functional programming, like Tupleware, but is a full relational DBMS. As such, their functional programming language LogiQL can be combined with declarative programming, and features a relational query optimizer. For optimizing the functional code, LogicBlox exploits constraint solving.

SimSQL [37] is another recent relational database with analytical features. Users write algorithms from scratch, which are then translated into SQL. Several SQL extensions, such as for-each style loops over relations, as well as vector and matrix data types, facilitate analytics in the database. While recognizing that its tuple-orientated approach to matrix-based problems results in low performance [36], SimSQL emphasizes its general-purpose approach. As a result of those design decisions, SimSQL is able to execute complex machine learning algorithms, which many other computation platforms are not able to [36], but lacks optimizations for standard analytical algorithms.

To conclude, while all presented database systems strive for integration of analytical and relational queries, the achieved level of integration vastly differs between systems. Most presented systems rely on black box execution of user-defined functions by the database, while others transform analytical into relational queries to allow for query inspection and optimization by the database.

5.3 HyPer For Data Analytics

HyPer [115] is a *hybrid* main-memory RDBMS that is optimized for both transactional and analytical workloads. It offers best-in-class performance for both workloads, even when operating simultaneously on the same data. Adding capabilities to execute data analysis algorithms is the next step towards a unified data management platform, without stale data warehouses.

Several features of HyPer contribute to its suitability for data analytics. First, Hyper *generates efficient data-centric code* from user queries, thus reducing the user's responsibility to write algorithms in an efficient way [149]. After transforming the query into an abstract syntax tree (AST), multiple optimization steps, and the final translation into a tree of physical operators, HyPer generates code using the LLVM compiler framework. Computation-intensive algorithms benefit from this design, as, e.g., function calls are omitted. As a result, users without knowledge in efficient algorithms can write fast analytical queries.

Second, performance is further improved by ensuring *data locality.* Data-centric execution attempts keeping data tuples in CPU registers as long as possible to avoid unnecessary copying of data. If possible, a tuple is therefore kept in registers while multiple operators are executed on it. This so-called *pipelining* is most important for queries that touch tuples multiple times. For ad-hoc analytical queries pre- and post-processing steps can be combined with the data processing to generate highly efficient machine code. As many analytical algorithms are pipeline breakers, in practice we pipeline pre-processing and data materialization as well as result generation and post-processing.

Third, HyPer focuses on *scale-up* on multi-core systems rather than on *scale-out* on clusters, hence, parallelization of the operators and the generated code is a performance-critical aspect. Characteristics of modern har[t]dware, such as non-uniform memory access (NUMA), cache hierarchies, and vector processing, therefore have to be taken into account when new features are integrated into the DBMS. Avoiding data distribution onto multiple nodes is especially important when the input data cannot be chunked easily, e.g., when processing graph-structured data.

Beside efficient integration of algorithms, other characteristics further encourage the use of a specific RDBMS for data analysis use-cases: HyPer provides a PostgreSQL-compatible SQL interface, it is fully ACID-compliant and offers fast data loading [143] which is especially important for data scientists.

5.4 In-database Processing

Existing systems for data analysis often use their own, proprietary query languages and APIs to specify algorithms, e.g., Apache Spark [205] and Apache Flink [13]. This approach has several drawbacks: Unusual query languages make it necessary to extensively train the domain experts that write queries. If common high-level programming languages, like Java, are used, many programmers are available, but they usually lack domain knowledge. Additionally, optimizing high-level code is a hard problem that compiler designers have been working on for decades, especially in combination with additional query execution logic.

Our goal is to enable data scientists to create and execute queries in a *straight-forward* way, while keeping all *flexibility* for expert users. In this chapter, we assess multiple approaches to integrate data analytics into HyPer. The first layer shown in 5.1, using the database system solely as data storage, is omitted here as it does not belong to the in-database processing category. Layers two and three—UDFs and SQL queries, respectively—are already implemented in various database systems. Layer four describes our novel approach of deeply integrating complex algorithms into the database core to maximize query performance while retaining flexibility for the user.

5.4.1 Program Execution within the Database

Many RDBMSs allow user-defined functions (UDFs), in which database users can add arbitrary functionality to the database. This eliminates the need to copy data to external systems. The code snippets are registered with the database system and are usually run by the database system as a black box, though first attempts to "open the black box" have been made [100]. If UDF code contains SQL queries, executing these queries potentially required costly communication with the database. This is because for most UDF languages it is not possible to bind together the black box code and the code that executes the embedded SQL query, hence, foregoing massive optimization potential. Because of the dangers to stability and security that go along with executing foreign code in the database core, a sandbox is required to separate database code and user code.

5.4.2 Extensions to SQL

There is general consensus that relational data should be queried using SQL. By extending SQL to integrate new algorithms, the vast amount of SQL infrastructure, e.g., JDBC connectors and SQL editors, can be reused to work with analytical queries. Furthermore, the declarative nature of SQL makes it easy to continuously introduce new optimizations. Also, using this common language, one avoids the high effort of creating a new language, and of teaching it to users.

Some algorithms, e.g., the a-priori algorithm [28] for frequent itemset mining, work well in SQL, but others are difficult to express in SQL and even harder to optimize. One common difficulty is the *iterative nature* of many analytical algorithms. To express iterations in SQL, recursive common table expressions (CTE) can be used. CTEs compute a monotonically growing relation, i.e., tuples of *all* previous iterations are kept. As many iterative algorithms need to access *one* previous iteration only, memory is wasted if the optimizer does not optimize this hard-to-detect situation. This is a problem especially for main-memory databases where memory is a scarce resource.

To solve this issue, we suggest an iteration concept for SQL that does not *append* to the prior iteration, but instead *replaces* it, and therefore drastically reduces the memory footprint of iterative computations. As the intermediate results become smaller, less data has to be read and processed, thus, it also improves analytics performance. We explain the details of our iteration concept in 5.5.

5.4.3 Data Analytics in the Database Core

In contrast to other database systems, HyPer additionally integrates important data analytics functionality directly into the core of the database system by implementing special highly-tuned *operators* for them. Because the internal structures of database systems are fairly different, such operators have to be specifically designed and implemented for each system. Differentiating factors between systems are, among others, the execution model (tuple-at-a-time vs. vectorized execution) as well as the storage model (row store vs. column store). For example, an operator in the analytical engine Tupleware, which does not support updating datasets, would look significantly different from an operator in the full-fledged database system HyPer, which needs to take care of updates and query isolation.

HyPer can arbitrarily mix relational and analytical operators, which leads to a seamless integration between analytics with other SQL statements into one query plan. This is especially useful because the functionality of existing relational operators can be reused for common subtasks in analytical algorithms, such as grouping or sorting. Thus, analytical operators can focus on optimizing the algorithms' core logic such as providing efficient internal data representations, performing algorithm-dependent pre-processing steps, and speeding up computation-intensive loops. A further advantage of custom-built analytical operators is that the query optimizer knows their exact properties and can choose an optimal query plan based on this information. Having all pre- and post-processing steps in one language—and one query—greatly simplifies data analytics and allows efficient ad-hoc queries. In 5.6 we elaborate on our implementation of (physical) operators.

Out of the integration layers presented in this section, special operators are integrated most deeply into the database. As a result, they provide unrivaled performance, but have the disadvantage of reducing the user's flexibility. To aid this and regain flexibility, we propose lambda expressions as a way to inject user-defined code into operators. Lambdas can, for example, be used specify distance metrics in the k-Means algorithm or to define edge weights in PageRank.

By implementing multiple of these layers, we can offer data analytics functionality to diverse user groups: User-defined algorithms are attractive for data scientists that want to implement specific algorithms in their favorite programming language without having to copy the data to another system. Persons knowledgeable in analytical algorithms and SQL might prefer to stick to their standard data querying language, hence, extensions to SQL are the best choice for them. Algorithm operators that are implemented by the database developers are targeted towards users that are familiar with the data domain but not with data analytics algorithm design.

Syntactically, UDFs, stored SQL queries and special operators cannot and should not be distinguished by the user. This way database system architects can, transparent to the user, decide on an algorithm's level of integration.

In the following sections, we delve into the details of data analytics using SQL and using specialized analytical operators with λ-expressions. We omit the details on the first two layers—using the database solely as data storage, and running UDFs in a black box within the database—because the first does not incorporate any analytical algorithms on the database system side and the latter uses the database system as a runtime environment for user-defined code. When the database is only used to provide the data, the performance is bound by data transfer performance and the data analytics software used to run the algorithms. In case the database is used to execute code in a black box, again, the runtime depends on the programming language and implementation used in these UDFs.

5.5 Data Analytics using SQL

Our overall goal is to seamlessly integrate analytical algorithms and SQL queries. In the third layer, which is described in this section, SQL is used and extended to achieve this goal. Standard SQL already provides most functionality necessary for implementing analytical algorithms, such as fix point recursion, aggregation, sorting, or distinction of cases. However, one vital construct is missing: A more general concept of iteration has to be added to the language. The following 5.5.1 introduces this general-purpose iteration construct, called *iterate operator*. Afterwards, query optimization for analytical queries is discussed in 5.5.2.

Our running example are the three algorithms k-Means, Naive Bayes, and PageRank which are well-known [193, 7] examples from vector and graph analytics and used as example building blocks in other state-of-the-art analytics systems [48]. Their properties are shared by many other data mining and graph analytics algorithms. Furthermore, they represent the areas of data mining, machine learning, and graph analytics, respectively. Thus, they are suiting examples for this chapter.

5.5.1 The Iterate Operator

The SQL:1999 standard contains recursive common table expressions (CTE) that are constructed using the `with recursive`. Recursive CTEs allow for computation of growing relations, e.g., transitive closures. In these queries, the CTE can be accessed from within its definition, and is iteratively computed until no new tuples are created in an iteration. In other words, until a fixpoint is reached. Although it is possible to use this fixpoint recursion concept for general-purpose iterations, this is clearly a diversion from its intended use case, and can thus result in wrong optimizer decisions.

Our iterate operator has similar capabilities as recursive CTEs:

Listing 5.1: Syntax of the *iterate* SQL language extension. A temporary table *iterate* is created, that in the beginning contains the result of the *initialization* subquery. Iteratively, the subquery *step* is applied to the result of the last iteration, until the boolean condition *stop condition* is fulfilled.

```
SELECT * FROM ITERATE([initialization], [step], [stop condition]);

-- find smallest three-digit multiple of seven
SELECT * FROM ITERATE((SELECT 7 "x"),
     (SELECT x+7 FROM iterate),
     (SELECT x FROM iterate WHERE x>=100));
```

It can reference a relation within its definition, hence, allowing for iterative computations.

In contrast to recursive CTEs, the iterate operator *replaces* the old intermediate relation rather than *appending* new tuples. Its final result is thus a table with the tuples that were computed in the last iteration only. This pattern is often used in data and graph mining algorithms, especially when some kind of metric or quality of data tuples is computed. In PageRank, for example, the initial ranks are updated in each iteration. In clustering algorithms, the assignment of data tuples to clusters has to be updated in every iteration. These algorithms have in common that they operate on fixed-size datasets, where only certain values (ranks, assigned clusters, et cetera) are updated. This means the stop criterion has to be changed: Rather than stopping when no new tuples are generated, our iterate operator stops when a user-defined predicate evaluates to true. We show the syntax of the iteration construct in 5.1. By providing a non-appending iteration concept with a while-loop-style stop criterion, we are adding more semantics to the implementation, which has been shown to massively speed up query execution due to better optimizer decisions [48].

Although it is possible to implement the afore-mentioned algorithms using recursive CTEs, the iterate operator has two major advantages:

- Lower memory consumption: Given a dataset with n tuples, and i iterations. With recursive CTEs, the table is growing to $n * i$ tuples. Using our operator, the size of the relation remains n. For comparisons of the current and the last iteration, we need to store $2*n$ tuples, and can discard all prior iterations early. Hence, the iterate construct saves vast amounts of memory in comparison to recursive CTEs. Furthermore, if the stop criterion is the number of executed iterations, recursive CTEs have to carry along an iteration counter, which is a huge memory overhead because it has to be stored in every tuple.

- Lower query response times: Because of the smaller relation size, our algorithm is also faster in scanning and processing the whole relation, which is necessary to re-compute the ranks, clusters, et cetera.

Lower memory requirements are particularly important in main-memory databases like HyPer, where memory is a scarce resource. This is especially true when whole algorithms are integrated into the database, because they often need additional temporary data structures. Our evaluation, 5.8, shows how algorithm performance can be improved by using our iterate operator instead of recursive CTEs, while keeping the flexibility of `with recursive` statements. Both approaches share one drawback, they can both produce infinite loops. Those situations need to be detected and aborted by the database system, e.g., via counting recursion depth or iterations, respectively.

A conceptually similar idea, that also features appending and non-appending iterations, can be found in the work of Binnig et al. [26]. Being a language proposal for a functional extension to SQL, this chapter does neither discuss where which type of iteration is appropriate, nor lists advantages and drawbacks regarding performance or memory consumption.

5.5.2 Query Optimization and Seamless Integration with the Surrounding SQL Query

Keeping intermediate results small by performing selections as early as possible is a basic principle of query optimization. This technique, called pushing selections, is in general not possible when analytical algorithms are affected. This is because the result of an analytical algorithm is not determined by *single tuples* (as it is for example for joins), but potentially influenced by the *whole* input dataset. A similar behavior can be found in the group-by operator, where the aggregated results also depend on *all* input tuples. This naturally narrows the search space of the query optimizer and reduces optimization potentials.

One mayor influencing factor for query optimization is the cardinality of intermediate results. For instance, precise cardinality estimations are necessary for choosing the best join ordering in a query. It is, however, hard to estimate the output cardinality of the generic iterate operator, because it can contain diverse algorithms. Some algorithms, e.g., k-Means, iterate over a given dataset, hence, the number of tuples stays the same before and after the iterate operator. Other algorithms, e.g., reachability computations, increase the dataset with each iteration, which makes the final cardinality hard to estimate. Cardinality estimation on recursive CTEs faces the same difficulty and thus, similar estimation techniques can be applied. To conclude, the diverse nature of analytical algorithms does not offer many generic optimization opportunities. Instead, relational query optimization has to be performed almost independently on the subqueries *below* and *above* the analytical algorithm, while the analytical algorithm itself might benefit from different optimization techniques, e.g., borrowed from general compiler design or constrained solving, as suggested by [17]. Because of the lacking potential for standard query optimization, low-level optimizations such as vectorization and data locality, as introduced in 5.3, become more important.

5.6 Operators

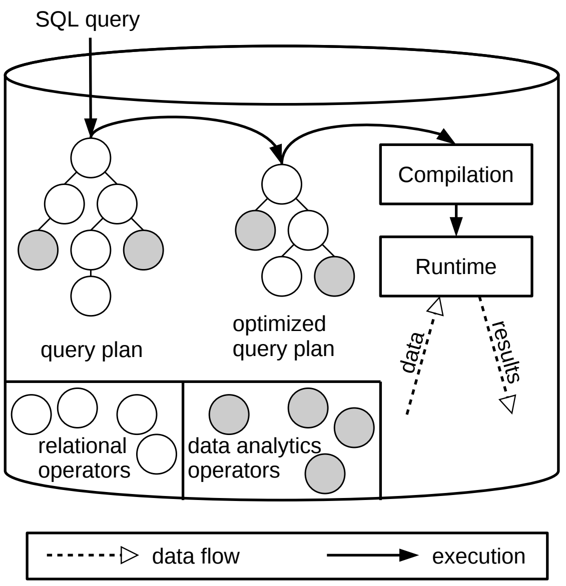


Figure 5.3: Query translation and execution with relational and analytical operators. A SQL query is translated to an abstract syntax tree (AST) consisting of both relational and analytical operators. The optimizer can inspect both types of operators. This approach provides highest integration and performance.

The most in-depth integration of analytical algorithms into a DBMS is by providing implementations in the form of physical operators. Physical operators like hash join or merge sort are highly optimized code fragments that are plugged together to compute the result of a query. All physical operators, including the analytical ones introduced in this chapter, use tables as input and output. They can be freely combined, ensuring maximal efficiency. 5.3 shows how physical analytics operators are integrated into query translation and execution. Physical operators are performance-wise superior to the general iteration construct, introduced in 5.5.1, as these specialized operators know invariants of their algorithms such as the estimated output cardinality, or data dependencies in complex computations. Thus, they know best how to distribute work among threads or how to optimize the memory layout of internal data structures.

Listing 5.2: Operator integration in SQL. Arbitrary preprocessing of input data and arbitrary post-processing of the results is possible. Additional parameters define the algorithm's behavior.

```
SELECT * FROM PAGERANK((SELECT src, dest FROM edges), 0.85, 0.0001);
```

For example, the query shown in 5.2 computes the PageRank value for every vertex of the graph induced by the *edges* relation[10]. The query is processed by a table scan operator followed by our specialized physical PageRank operator. The PageRank operator implementation defines, e.g., whether parallel input (from the table scan operator) can be processed, an information that is used by the optimizer to create the best plan for the given query.

In the next sections, we describe the chosen algorithms, k-Means, Naive Bayes, and PageRank, and how we implemented them in HyPer. Furthermore, we describe necessary changes to the optimizer.

5.6.1 The Physical k-Means Operator

k-Means is a fast, model-based iterative clustering algorithm, i.e., it divides a set of tuples into k spherical groups such that the sum of distances is minimized. It can be utilized as building block for advanced clustering techniques. The classical k-Means algorithm by Lloyd [131] splits each iteration into two steps: assignment and update. In the assignment step, each tuple is assigned to the nearest cluster center. In the update step, the cluster centers are set to be the arithmetic mean of all tuples assigned to the cluster. The algorithm converges when no tuple changes its assigned cluster during an iteration. For practical use, the convergence criterion is often softened: Either, a maximum number of iterations is given, or the algorithm is interrupted if only a small fraction of tuples changed its assigned cluster in an iteration.

In our implementation, the k-Means operator requires two input relations—data and initial centers—that are passed via subqueries. An additional parameter defines the maximum number of iterations. Using parallelism, our implementation benefits from modern multi-core systems: Each thread locally assigns data tuples to their

[10]Parentheses around the subquery are necessary because arbitrary queries are allowed there. The sole use of commas would have lead to an ambiguous grammar.

nearest center and to prepare the re-computation of cluster centers, each thread sums up the tuples values. The data tuples itself are consumed and directly thrown away after processing. For the next iteration, tuples are requested again from the underlying subquery. As a result, the query optimizer can decide to compute the data relation each time, or use materialized intermediate results, whatever is faster in the given query. Data locality is ensured because all centers and intermediate data structures are copied for each thread. Thread synchronization is only needed for the very last steps, global aggregation of the local intermediate results and the final update of the cluster centers. This procedure is repeated until the solution remains stable (i.e., no tuple changed its assignment during an iteration), or until the maximum number of iterations is reached. The operator then outputs the cluster centers.

5.6.2 The Physical Naive Bayes Operator

Naive Bayes classification aims at classifying entities, i.e., assigning categorical labels to them. Other than k-Means or PageRank, it is a supervised algorithm, and, thus, consists of two steps performed on two different datasets: First, a dataset A with known labels is used to build a model based on the Bayesian probability density function. Second, the model is applied to a related but un-labeled (thus, unknown) dataset B to predict its labels. When implemented in a relational database, one challenge is storing the model, as it does not match any of the relational entities—relation, index—completely.

We implemented model creation and application as two separate operators, Naive Bayes training and Naive Bayes testing, respectively. The generation of additional statistical measures is handled by two additional operators, that are not limited to Naive Bayes but can be used as building block for multiple algorithms, for example k-Means.

Similar to k-Means, the Naive Bayes training operator is a pipeline breaker. Each threads holds a hash table to manage its input data with the class as key, while not storing the tuples itself. In addition, the number of tuples N is stored for each class, as well as the sum of the values $\sum_{n \in N} n.a$ and the sum of the square of each value $\sum_{n \in N} na^2$ for each class and attribute. After the whole input is consumed, the training operator computes the a-priori probability for each class as well as the mean and standard deviation for each class and attribute:

For a given training set D with N instances $n \in D$ which contains a set of classes C with $c \in C$ being a single class and $|c|$ the number of instances of this class c in D. Then, the a-priori probability of a class is given by:

$$PR(c) = \frac{|c| + 1}{N + |C|}$$

5.6.3 The Physical PageRank Operator

PageRank [34] is a well-known iterative ranking algorithm for graph-structured data. Each vertex v in the graph, e.g., a websites or person, is assigned a ranking value that can be interpreted as its importance. The rank of v depends on the number and rank of incoming edges, i.e., v is important if many important vertices have edges to it. A PageRank iteration is a sparse matrix-vector multiplication. In each iteration, part of each vertex's importance flows off to the vertices it is adjacent to, and in turn each vertex receives importance from its neighbors. Similar to k-Means, PageRank converges towards a fixpoint, i.e., the vertex ranks change less than a user-defined epsilon. Also, it is common to specify a maximum number of iterations.

The sparse matrix-vector multiplication performed in the PageRank iterations is similar to many graph algorithms in that its performance greatly benefits from efficient neighbor traversals. This means for a given vertex v it has to be efficiently possible to enumerate all of its neighbors. Our PageRank implementation ensures this by efficiently creating a temporary compressed sparse row (CSR) representation [182] that is optimized for the query at hand. We avoid storage overhead and an access indirection in this mapping by re-labeling all vertices and doing a direct mapping. After the PageRank computation we use a reverse mapping operator that translates our internal vertex ids back to the original ids.

The PageRank operator uses only the CSR graph index and does not need to access the base data anymore. In each iteration we compute the vertices' new PageRank values in parallel without any synchronization. Because we have dense internal vertex ids we are able to store the current and last iteration's rank in arrays that can be directly indexed. Thus, every neighbor rank access only involves a single read. At the end of each iteration we aggregate each worker's data to determine how much the new ranks differ from the previous iterations. If the difference is less or equal to the user-defined epsilon or if the maximum iteration count is reached, the PageRank computation finishes.

5.7 Lambda Expressions

In 5.4.3 we describe the integration of specialized data analytics operators into the database core. These operators provide unrivaled performance in executing the algorithms they were designed for. However, without modification they are not flexible, i.e., they are not even applicable in the context of similar but slightly different algorithms. Consider the k-Medians algorithm. It is a variant of k-Means that uses the L1-norm (Manhattan distance) rather than the L2-norm (Euclidean distance) as distance metric. While this distance metric differs between the variants, their implementations have in common predominant parts of their code. Even though this common code could be shared, different metrics would make necessary different variants of our algorithm operators.

Instead, when designing data analytics operators, we identify and aim to exploit such similarities. Our goal is to have one operator for a whole class of algorithms with variation points that can be specified by the user. To inject user-defined code into variation points of analytics operators we propose using *lambda expressions* in SQL queries.

Lambda expressions are anonymous SQL functions that can be specified inside the query. For syntactic convenience, the lambda expressions' input and output data types are automatically inferred by the database system. Also, for all variation points we provide default lambdas that are used should none be specified. Thus, non-expert users can easily fall back to basic algorithms. With lambda-enabled operators we strive not only to keep implementation and maintenance costs low, but especially to offer a wide variety of algorithm variants required by data scientists. Also, because lambda functions are specified in SQL, they benefit from existing relational optimizations.

5.3 show how our k-Means operator benefits from lambdas. In the `kmeans` function call's third argument, a lambda expression is used to specify an arbitrary distance metric. The operator expects a lambda function that takes two tuple variables as input arguments and returns a (scalar) float value. At runtime, these variables are bound with the corresponding input tuples to compute the distance. Thus, by providing a k-Means operator that accepts lambda expressions we do

Listing 5.3: Customization of the k-Means operator using a lambda expression for the distance function.

```
CREATE TABLE data (x FLOAT, y INTEGER, z FLOAT,
                   desc VARCHAR(500));
CREATE TABLE center (x FLOAT, y INTEGER, z FLOAT);
INSERT INTO data ...
INSERT INTO center ...

SELECT * FROM KMEANS(
  -- sub-queries project the attributes of interest
  (SELECT x,y FROM data),
  (SELECT x,y FROM center),
  -- the distance function is specified as λ-expression
  λ(a,b) (a.x-b.x)^2+(a.y-b.y)^2,
  -- termination criterion: max. number of iterations
  3
);
```

not only cover the common k-Means and k-Medians algorithms but also allow users to design algorithms that are specific to their task and data at hand. These custom algorithms are still executed by our highly-tuned in-database operator implementation and because all code is compiled together, no virtual function calls are involved.

5.8 Experimental Evaluation

In this section, we evaluate our implementations of k-Means, PageRank, and Naive Bayes. As introduced in 5.4, we implemented multiple versions of the algorithms, that reflect different depths of integration. We compare our solutions to other systems that are commonly used among data scientists. This includes middle-ware tools based on RDBMSs, analytics software for distributed systems, and standalone data analysis tools.

5.8.1 Datasets and Parameters

We use a variety of datasets to evaluate the influence of certain characteristics of the datasets and workload to the resulting performance.

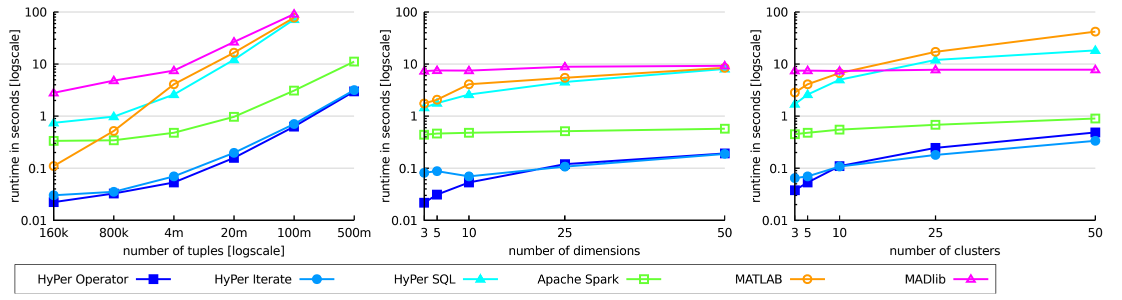


Figure 5.4: k-Means experiments. From left to right: varying the number of tuples N, dimensions d, and clusters k. Default parameters: 4,000,000 tuples, 10 dimensions, 5 clusters, 3 iterations.

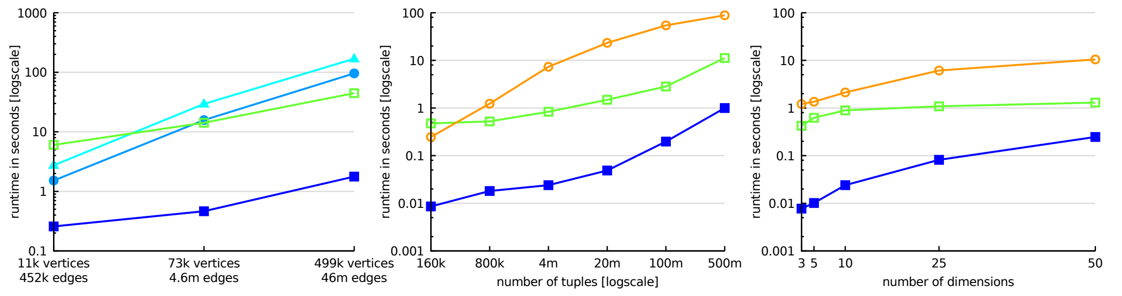


Figure 5.5: PageRank and Naive Bayes experiments. From left to right: PageRank using the LDBC SNB dataset, dumping factor 0.85, and 45 iterations. Naive Bayes experiment varying the number of tuples N. Naive Bayes experiment varying the number of dimensions d.

k-Means Datasets and Parameters

k-Means is an algorithm targeted at *vector data*, i.e., tuples with a number of dimensions. This data model fits perfectly into relations. The data is characterized by the number of tuples n, the number of dimensions d used for clustering, and the data types of the dimensions. We chose to perform experiments for varied n and d, while keeping the data types constant. In addition to the dataset, also the algorithm itself has multiple parameters: the number of clusters k, the cluster initialization strategy, and the number of iterations i that are computed. The number of clusters k drastically influences the query performance because it defines the number of distances that have to be computed and compared, and is, hence, an important parameter in our evaluation. To produce comparable results with

Table 5.1: Datasets for k-Means experiments.

	#tuples n	**#dimensions** d	k
Varying number of tuples	160 000	10	5
	800 000	10	5
	4 000 000	10	$5^{\star}$
	20 000 000	10	5
	100 000 000	10	5
	500 000 000	10	5
Varying number of dimensions	4 000 000	3	5
	4 000 000	5	5
	4 000 000	10	$5^{\star}$
	4 000 000	25	5
	4 000 000	50	5
Varying number of clusters	4 000 000	10	3
	4 000 000	10	$5^{\star}$
	4 000 000	10	10
	4 000 000	10	25
	4 000 000	10	50

$\star$ same experiments, for connecting the three lines of experiments

a wide range of systems, our experiments use the simplest cluster initialization strategy: random selection of k initial cluster centers. We chose to perform three iterations i. This keeps the experiment duration short while leveling out a possible overhead in the first iteration.

While modifying one parameter, we keep the other two fixed, to focus on the effect on that parameter only. The resulting list of experiments is shown in 5.1. We conduct five to six experiments per parameters, which allows us to assess not only the performance, but also the scaling behavior of the different systems. The dataset sizes—determined by n and d—were chosen to be processable by all evaluated systems within main memory and within a reasonable time, given the vast performance differences between the systems. We create artificial—uniformly distributed—datasets because they provide an important advantage over real-world datasets in our use case: As the performance of plain k-Means with a fixed number of iterations is irrespective of data skew, our decision to use synthetic datasets does not introduce any drawbacks.

Naive Bayes Datasets and Parameters

The Naive Bayes experiments are conducted using the same synthetic datasets as k-Means. We vary the number of tuples N and the number of dimensions d. For the labels we chose a uniform probability density function of two labels 0 and 1. Our experiments cover the *training* phase of the algorithm only, as it has a much higher complexity and thus runtime than the *testing* step.

PageRank Datasets and Parameters

PageRank is an algorithm targeted at *graph data*, i.e., vertices and edges with optional properties. The algorithm is parameterized with the damping factor d, modeling the probability that an edge is traversed, e, the maximum change between two iterations for which the computation continues, and the maximum number of iterations i. For the damping factor d, we chose the reasonable value 0.85 [34], i.e., the modeled random surfer continues browsing with a probability of 85%. To better compare different systems, we set e to 0 and run a fixed number of 45 iterations in all systems. As datasets we use the artificial LDBC graph that is designed to follow the properties of real-world social networks. We generated multiple LDBC graph in different sizes up to 500,000 vertices and 46 million edges, using the SNB data generator [60], and used the resulting undirected person-knows-person graph.

5.8.2 Evaluated Systems

We evaluate our physical operators, denoted as *HyPer Operator*, SQL queries with our iterate operator, denoted as *HyPer Iterate*, and a pure SQL implementation using recursive CTEs, denoted as *HyPer SQL*, against diverse data analysis systems introduced in 5.2. We chose *MATLAB R2015* as a representative of the "programming languages" group. The next category are "big data analytics" platforms: We evaluate *Apache Spark 1.5.0* with MLlib. As contender in the "database extensions" category, we chose *MADlib 1.8* on top of the Pivotal Greenplum Database 4.3.7.1.

To ensure a fair comparison, all systems have to implement the same variant of k-Means: Lloyd's algorithm. Note that we therefore disabled the following optimizations implemented in Apache Spark MLlib: First, the MLlib implementation computes lower bounds for distances using norms, hence, reducing the number of distance computations. Second, distance computation uses previously computed norms instead of computing the Euclidean distance (if the error introduced by this method is not too big). *litekmeans*[11], a fast k-Means implementation for MATLAB, uses the same optimizations. We therefore use MATLAB's built-in k-Means implementation in our experiments.

5.8.3 Evaluation Machine

All experiments are carried out on a 4-socket Intel Xeon E7-4870 v2 (15×2.3 GHz per socket) server with 1 TB main memory, running Ubuntu Linux 15.10 using kernel version 4.2. Greenplum, the database used for MADlib, is only available for Red Hat-based operating systems. We therefore set up a Docker container running CentOS 7. The potential introduced overhead is considered in our discussion. As mentioned, we chose the datasets to fit into main memory, even when considering additional data structures. MATLAB does not contain parallel versions of the chosen algorithms, as mentioned in 5.2. This issue is also considered in our result discussion.

5.8.4 Results and Discussion

Figures 5.4 and 5.5 display the total runtimes measured in our experiments. In general, the results match our claims regarding the four layers[t] of integration and their runtimes as shown in 5.1: Systems using UDFs (layer 2)—in our experiments represented by MADlib—are slower than HyPer Iterate and HyPer SQL, using SQL (layer 3). The fastest implementation—HyPer Operator—uses analytical operators (layer 4). Runtime of dedicated analytical systems, such as MATLAB and Apache Spark, heavily depends on the individual system.

Recursive CTEs and HyPer Iterate

As claimed in 5.5, using the iteration concept improves runtimes over plain SQL. While the pure SQL implementation, using recursive CTEs, has to store and process intermediate results that grow with each iteration, the iteration operator's intermediate results have constant size. In our implementations, this means

[11]`http://www.cad.zju.edu.cn/home/dengcai/Data/Clustering.html`

additional selection predicates for the pure SQL variant, and more expensive aggregates due to the larger intermediate results.

Hyper Operator and HyPer Iterate

The k-Means experiments show almost no difference between the HyPer Operator and the HyPer Iterate approach. k-Means is a rather simple algorithm: there is no random data access, only few branches, vectorization can be applied easily, and the data structures are straightforward. Furthermore, it operates on vector data, thus, both operator and SQL implementations use similar internal data structures. This results in very similar code being generated by the operator and the query optimizer, hence, the similar runtimes.

For PageRank, the experiments reveal a different picture: HyPer Operator runs significantly faster than HyPer Iterate because of its optimized CSR graph data structure. In contrast, HyPer Iterate has to work on relational structures—an edges table and a derived vertices table—and, thus, needs to perform many (hash) joins. As a result, its runtime is dominated by building and probing hash tables. This behavior is also found in [106], where a SQL implementation of PageRank also showed performance only comparable to stand-alone-systems. The following rule of thumb can be applied: The more similar optimized SQL code and code generated from the hand-written operator are, the smaller the runtime difference between HyPer Iterate and HyPer Operator approach.

HyPer, MATLAB, MADlib, and Apache Spark

Among the contender systems, Apache Spark shows by far the best runtimes, which was expected because Spark was especially built for these kinds of algorithms. Still, it is multiple times slower than our HyPer Operator approach for all three evaluated algorithms, as shown in the Figures 5.4 and 5.5. HyPer's one-system-fits-all approach comes with some overhead of database-specific features that are not present in dedicated analytical systems like Apache Spark. Therefore, it is important that these features do not cause overhead when they are not used. For instance, isolation of parallel transactions should not take a significant amount of time when only one analytical query is running. Some database-specific overhead, stemming, e.g., from memory management and user rights management, cannot be avoided. Nevertheless, HyPer shows far better runtimes than dedicated systems, while also avoiding data copying and stale data. MATLAB runs both algorithms single-threaded and therefore cannot compete, but was included because multiple heavily used data analytics tools do not support parallelism. MADlib, even taking into account the runtime impairment caused by the virtualization overhead, cannot compete with solutions that integrate data analytics deeper and, hence, produce better execution code.

Interestingly, Spark and MADlib seem to be almost not affected by the number of dimensions or clusters in the experiments. As algorithm-wise more complex computations are necessary if either of this numbers rises, we suspect those computations to be hidden behind multi-threading overhead. For example, if each thread handles one cluster, even the 50 clusters in the largest experiment still fit into the 120 hyper-threads of the evaluation machine. But k-Means with larger number of dimensions or clusters is not common, because their results are impaired by the curse of dimensionality or cannot be interpreted by humans, respectively. Regarding the scaling for larger datasets, log-scaled runtimes fail to show runtime differences appropriately: Plots with log-scaled runtimes counter-intuitively show converging lines when in fact the runtime difference between two systems is constant, which is the case for HyPer Operator/Iterate and Apache Spark in the leftmost k-Means figure.

The results presented above support our claim that a multi-layer approach helps targeting diverse user groups. DBMS manufacturers benefit from the identical interface and syntax of UDFs, stored SQL queries, and hard-coded operators. The decision in which layer an algorithm is implemented is, thus, solely affected by implementation effort versus gain in performance and flexibility. Laypersons can use these manufacturer-provided data analytics algorithms without having to care whether it is a UDF, an SQL query or a physical operator. Database users with expertise, opposed to laypersons, that want to implement their own analytical algorithms, can choose to implement either UDFs or SQL queries.

To put it in a nutshell, the experiments match the expected order concerning runtimes: the deeper the integration of data analytics, the faster the system. Our results also support our idea of one database system being sufficient for multiple workloads. While this has been shown for combining OLTP and OLAP workloads before [66, 115], our contribution was to integrate one more workload, data analytics, while keeping performance and usability on a high level.

5.9 Conclusion

We described multiple approaches of integrating data analytics into our relational main memory database system HyPer. Like most database systems, HyPer can be used as a data store for external tools. However, doing so exposes data transfer as a bottleneck and prevents significant query optimizations. Instead, we presented three layers of integrating data analytics directly into the database system: data analytics *in UDFs*, data analytics *in SQL*, and analytical *operators in the database core*. The layers' depth of integration and, hence, their analytics performance increases with each layer.

UDFs allow the user to implement arbitrary computations directly in the database. However, because the database runs UDFs as a black box, automated optimization potentials are very limited. To aid this lack of optimization potential, we proposed doing data analytics in SQL. As iterations are hard to express in SQL and difficult to optimize we presented the *iteration operator* and a corresponding language extension that serves as a building block for arbitrary iterative algorithms directly in SQL.

For major analytical algorithms that are used frequently, e.g., k-Means, PageRank, and Naive Bayes, we then proposed an even deeper integration: integrating highly-tuned analytical operators into the database core. Using our novel SQL *lambda expressions*, users can specialize analytical operators directly within their SQL queries. This adds flexibility to otherwise fixed operators and allows, for example, for applying arbitrary user-defined distance metrics in our tuned k-Means operator. Just like the iterate operator and the analytics operator, lambda expressions are part of the logical query plan and, thus, subject to query optimization and code generation. Hence, they benefit from decades of research in database systems.

Our presented approaches enable complete integration of data analytics in SQL queries, ensuring both efficient query plans and usability. In our experiments we saw that in HyPer data analytics on both graph and vector data is significantly faster than in dedicated state-of-the-art data analytics systems—e.g., 92 times faster than Apache Spark for PageRank. This is especially significant because, as an ACID-compliant database, HyPer must also be able to handle concurrent transactional workloads. Thus, we showed that HyPer is suitable for integrated data management *and* data analytics on large data, with multiple interfaces targeted at different user groups.

CHAPTER 6

Conclusion

In this thesis, we investigated in the challenges of creating data mining algorithms for heterogeneous data and showed how efficient integration of such methods in relational database systems would be possible. We focused on improving shortcomings of existing techniques and proposed more efficient methods for handling large vector-, graph- and spatio-temporal data.

We started by proposing the two methods NORD and ISAAC for subspace clustering, trying to erase the problem of insignificant results due to high redundancy with automized approaches. With NORD we managed to balance redundancy for axis parallel clusterings, while ISAAC extends NORD for arbitrary clusterings applying the ISA dimensionality reduction to find statistically independent subspaces. Both approaches make their own coding scheme basing in minimum description length [170] to erase hard to set input parameters for the user. We demonstrated the quality of both methods on synthetic and real data sets.

Next, we laid our focus on mining large graph data. We elaborated two methods, Cxprime and IROC to deal with the community detection in undirected graphs and in attributed graphs. Cxprime tries to find the communities via different substructures (cliques and stars)efficiently on sparse graphs. It again uses minimum description length for eradicating input parameters and assessing the quality of the found clustering automatically. Besides it is able to predict links in the graph structure. IROC finds overlapping communities on attributed graphs combining both the network structure with the relational vector structure of the attributes instead of clustering them separately. Here, the problem of redundancy applies to graphs as well but was solved due to a specific quality function of the minimum description length. But this raised another problem: The runtime degraded strongly due to the high complexity of the data structure, as such a tensor factorization approach TF-IROC was created to speed up the algorithm, but with that the automization was lost.

In order to expand our vector data sortiment, we proposed a spatio-temporal framework on the large MERRA weather data of NASA. The goal is to predict natural hazards in such large environmental data. For that this frameworks has three steps: First, it applies outlier detection on a given set that is later transformed into sparse tensors. Then in the second step these sparse tensors are factorized using a highly parallelizable PARAFAC tensor factorization method for dimensionality reduction. Then in the third step we classify the tensors with an ensemble of techniques. We demonstrated its performance and quality outcomes on the given real data sets that beated the state of the art method and discussed future directions.

Finally, we proposed how to integrate such and other relevant methods in relational database systems efficiently by making the databases optimizer work to support the given algorithm in SQL. We introduced the concept of analytics operators that make such optimizer-centric behavior possible and discussed why no specific (procedural) query language other than SQL should be used in that scenario. Besides we introduced so called lambda-functions that serve as parameter settings for such data mining operators in a database. Lambdas enable the possibility to generate strong semantical differences in a given data mining approach by selecting distance metrics an other important values before running the operator.

In conclusion, we proposed several methods including a full framework for efficient clustering, outlier detection and classification algorithms on various heterogeneous data types and use cases. Our ideas evolve around applying minimum description length for better quality functions and automizing approaches, and making special use of tensor factorization for runtime efficiency. Our framework was specifically created for the use case of large weather data and as such creates an own neighborhood-related way to find the outliers and then use it for building sparse tensors. Besides, integrating such methods as operators into RDBMS was very successful and dynamic approaches like the lambda functions can be further evaluated on their semantic differences. With these findings and application oriented approaches we hope that we can make a little impact in the future of data science.

Bibliography

[1] H. Abdi and L. J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[2] E. Acar, T. G. Kolda, and D. M. Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. *arXiv preprint arXiv:1105.3422*, 2011.

[3] E. Achtert, C. Böhm, J. David, P. Kröger, and A. Zimek. Global correlation clustering based on the hough transform. *Statistical Analysis and Data Mining*, 1(3):111–127, 2008.

[4] E. Achtert, C. Böhm, H.-P. Kriegel, P. Kröger, and A. Zimek. On exploring complex relationships of correlation clusters. In *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on*, pages 7–7. IEEE, 2007.

[5] E. Achtert, C. Böhm, H.-P. Kriegel, P. Kröger, and A. Zimek. Robust, complete, and efficient correlation clustering. In *SDM*. SIAM, 2007.

[6] E. Achtert, S. Goldhofer, H.-P. Kriegel, E. Schubert, and A. Zimek. Evaluation of clusterings–metrics and visual support. In *ICDE*, pages 1285–1288. IEEE, 2012.

[7] C. C. Aggarwal and H. Wang. Graph Data Management and Mining: A Survey of Algorithms and Applications. In C. C. Aggarwal and H. Wang, editors, *Managing and Mining Graph Data*, Advances in Database Systems, pages 13–68. Springer US, 2010.

[8] C. C. Aggarwal and P. S. Yu. *Finding generalized projected clusters in high dimensional spaces*, volume 29. ACM, 2000.

[9] N. Aggarwal, A. Kumar, H. Khatter, and V. Aggarwal. Analysis the effect of data mining techniques on database. *Advances in Engineering Software*, 47(1):164–169, 2012.

[10] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conference*, pages 94–105, 1998.

[11] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications*, volume 27. ACM, 1998.

[12] L. Akoglu, H. Tong, B. Meeder, and C. Faloutsos. Pics: Parameter-free identification of cohesive subgroups in large attributed graphs. In *SDM*, pages 439–450, 2012.

[13] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke. The stratosphere platform for big data analytics. *VLDBJ*, pages 939–964, 2014.

[14] O. Alter, P. O. Brown, and D. Botstein. Singular value decomposition for genome-wide expression data processing and modeling. 97(18):10101–10106, 2000.

[15] S. Alvarez and M. Vanrell. Texton theory revisited: A bag-of-words approach to combine textons. *Pattern Recognition*, 45(12):4312–4325, 2012.

[16] M. Ankerst, C. Elsen, M. Ester, and H.-P. Kriegel. Visual classification: An interactive approach to decision tree construction. pages 392–396, 1999.

[17] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. Design and implementation of the logicblox system. In *Proc. SIGMOD 2015*, pages 1371–1382. ACM, 2015.

[18] I. Assent, R. Krieger, E. Müller, and T. Seidl. Inscy: Indexing subspace clusters with in-process-removal of redundancy. In *ICDM Conference*, pages 719–724, 2008.

[19] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[20] E. Bae and J. Bailey. Coala: A novel approach for the extraction of an alternate clustering of high quality and high dissimilarity. In *ICDM*, pages 53–62. IEEE, 2006.

[21] A. Baraldi and P. Blonda. A survey of fuzzy clustering algorithms for pattern recognition. ii. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(6):786 –801, dec 1999.

[22] C. Baumgartner and D. Baumgartner. Biomarker discovery, disease classification, and similarity query processing on high-throughput ms/ms data of inborn errors of metabolism. *J Biomol Screen*, (11):90–99, 2006.

[23] C. Baumgartner, C. Plant, K. Kailing, H.-P. Kriegel, and P. Kröger. Subspace selection for clustering high-dimensional data. In *ICDM*, pages 11–18, 2004.

[24] S. Berchtold, C. Böhm, and H.-P. Kriegel. The Pyramid Technique: Towards breaking the curse of dimensionality. pages 142–153, 1998.

[25] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, pages 359–370, 1994.

[26] C. Binnig, R. Rehrmann, F. Faerber, and R. Riewe. FunSQL: It is time to make SQL functional. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 41–46. ACM, 2012.

[27] Bisk. What is big data? 2013.

[28] F. Bodon. A fast apriori implementation. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, 2003.

[29] C. Böhm, C. Faloutsos, J.-Y. Pan, and C. Plant. Robust information-theoretic clustering. In *KDD*, pages 65–75, 2006.

[30] C. Böhm, K. Kailing, P. Kröger, and A. Zimek. Computing clusters of correlation connected objects. In *SIGMOD*, pages 455–466. ACM, 2004.

[31] C. Böhm, C. Plant, J. Shao, and Q. Yang. Clustering by synchronization. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 583–592. ACM, 2010.

[32] P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *WWW*, pages 595–602, 2004.

[33] S. Botti, G. Dipoppa, A. Puiu, and S. Almaviva. Raman spectra massive classification using artificial neural networks. 2014.

[34] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Seventh International World-Wide Web Conference (WWW 1998)*, pages 3825–3833, 1998.

[35] L. Cai and T. Hofmann. Hierarchical document categorization with support vector machines. pages 78–87, 2004.

[36] Z. Cai, Z. J. Gao, S. Luo, L. L. Perez, Z. Vagena, and C. Jermaine. A comparison of platforms for implementing and running very large scale machine learning algorithms. In *Proc. SIGMOD 2014*, pages 1371–1382. ACM, 2014.

[37] Z. Cai, Z. Vagena, L. Perez, S. Arumugam, P. J. Haas, and C. Jermaine. Simulation of database-valued markov chains using simsql. In *Proc. SIGMOD 2013*, pages 637–648. ACM, 2013.

[38] D. Calandriello, G. muNiu, and M. Sugiyama. Semi-supervised information-maximization clustering. *CoRR*, abs/1304.8020, 2013.

[39] J. Cardoso. Multidimensional independent component analysis. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 4, pages 1941–1944. IEEE, 1998.

[40] D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1), 2006.

[41] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos. Fully automatic cross-associations. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 79–88, 2004.

[42] W.-C. Chang. On using principal components before separating a mixture of two multivariate normal distributions. 32(3):267–275, 1983.

[43] M. Chen, S. Mao, and Y. Liu. Big data: a survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.

[44] C.-H. Cheng, A. W. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *SIGKDD*, pages 84–93. ACM, 1999.

[45] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, pages 219–228, 2009.

[46] Y.-H. Chu, Y.-J. Chen, D.-N. Yang, and M.-S. Chen. Reducing redundancy in subspace clustering. *IEEE Transactions on Knowledge and Data Engineering*, 21(10):1432–1446, 2009.

[47] J. Costeira and T. Kanade. A multibody factorization method for independently moving objects. *International Journal of Computer Vision*, 29(3):159–179, 1998.

[48] A. Crotty, A. Galakatos, K. Dursun, T. Kraska, C. Binnig, U. Cetintemel, and S. Zdonik. An Architecture for Compiling UDF-centric Workflows. *Proc. VLDB Endow.*, 8(12):1466–1477, 2015.

[49] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. The case for interactive data exploration accelerators (ideas). In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 11:1–11:6. ACM, 2016.

[50] Y. Cui, X. Fern, and J. Dy. Non-redundant multi-view clustering via orthogonalization. In *ICDM 2007*, pages 133–142, Oct 2007.

[51] Y. Cui, X. Z. Fern, and J. G. Dy. Non-redundant multi-view clustering via orthogonalization. In *ICDM*, pages 133–142. IEEE, 2007.

[52] X.-H. Dang and J. Bailey. A hierarchical information theoretic technique for the discovery of non linear alternative clusterings. In *SIGKDD*, pages 573–582. ACM, 2010.

[53] M. Das and S. Parthasarathy. Anomaly detection and spatio-temporal analysis of global climate system. In *SensorKDD '09*, 2009.

[54] T. H. Davenport and D. Patil. Data scientist. *Harvard business review*, 90:70–76, 2012.

[55] G. R. T. de Lima and S. Stephany. A new classification approach for detecting severe weather patterns. *Computers & Geosciences*, 57:158 – 165, 2013.

[56] M. DeMaria and J. Kaplan. A statistical hurricane intensity prediction scheme (ships) for the atlantic basin. *Weather and Forecasting*, 9(2):209–220, 1994.

[57] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 89–98, New York, NY, USA, 2003. ACM.

[58] C. H. Q. Ding, X. He, H. Zhab, M. Gu, and H. D. Simon. A Min-max Cut Algorithm for Graph Partitioning and Data Clustering. In *IEEE International Conference on Data Mining*, pages 107–114, 2001.

[59] E. Elhamifar and R. Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *CoRR*, abs/1203.1005, 2012.

[60] O. Erling, A. Averbuch, J. Larriba-Pey, H. Chafi, A. Gubichev, A. Prat, M.-D. Pham, and P. Boncz. The LDBC Social Network Benchmark: Interactive Workload. In *Proc. SIGMOD 2015*, pages 619–630. ACM, 2015.

[61] M. Ester, H.-P. Kriegel, and J. Sander. Knowledge discovery in spatial databases. pages 61–74, 1999.

[62] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD Conference*, 1996.

[63] M. Ester and J. Sander. *Knowledge Discovery in Databases: Techniken und Anwendungen.* 2000.

[64] J. H. Faghmous and V. Kumar. *Spatio-temporal Data Mining for Climate Data: Advances, Challenges, and Opportunities*, pages 83–116. Springer Berlin Heidelberg, 2014.

[65] L. Faivishevsky and J. Goldberger. Nonparametric information theoretic clustering algorithm. In *ICML*, pages 351–358, 2010.

[66] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[67] U. M. Fayyad, G. P. Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* 1996.

[68] J. Feng, X. He, N. Hubig, C. Böhm, and C. Plant. Compression-based graph mining exploiting structure primitives. In *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, 2013.

[69] J. Feng, X. He, B. Konte, C. Böhm, and C. Plant. Summarization-based mining bipartite graphs. In *KDD*, pages 1249–1257, 2012.

[70] D. Fisher. Handbook of data mining and knowledge discovery. chapter Data mining tasks and methods: Clustering: conceptual clustering, pages 388–396. Oxford University Press, Inc., New York, NY, USA, 2002.

[71] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In *Knowledge Discovery in Databases*, pages 1–30. AAAI/MIT Press, 1991.

[72] A. R. Ganguly and K. Steinhaeuser. Data mining for climate change and impacts. *ICDMW '08*, pages 385–394.

[73] F. I. T. V. Gergely Palla, Derenyi Imre. Uncovering the overlapping community structure of complex networks in nature and society. *Nature Letter*, abs/1010.1523:814–818, 2005.

[74] J.-M. Geusebroek, G. J. Burghouts, and A. W. Smeulders. The amsterdam library of object images. *International Journal of Computer Vision*, 61(1):103–112, 2005.

[75] D. Gondek and T. Hofmann. Non-redundant data clustering. In *ICDM*, pages 75–82. IEEE Computer Society, 2004.

[76] D. Gondek and T. Hofmann. Non-redundant data clustering. *Knowledge and Information Systems*, 12(1):1–24, 2007.

[77] S. Gregory. *PKDD 2007*, chapter An Algorithm to Find Overlapping Community Structure in Networks, pages 91–102.

[78] S. Gregory. A fast algorithm to find overlapping communities in networks. In *ECML/PKDD (1)*, pages 408–423, 2008.

[79] P. Gruber, H. W. Gutch, and F. J. Theis. Hierarchical extraction of independent subspaces of unknown dimensions. In *Independent Component Analysis and Signal Separation*, pages 259–266. Springer, 2009.

[80] P. Grünwald. *The Minimum Description Length Principle.* MIT Press, Cambridge, MA, 2007.

[81] S. Günnemann, B. Boden, and T. Seidl. Db-csc: A density-based approach for subspace clustering in graphs with feature vectors. In *ECML/PKDD (1)*, pages 565–580, 2011.

[82] S. Günnemann, I. Färber, B. Boden, and T. Seidl. Subspace clustering meets dense subgraph mining: A synthesis of two paradigms. In *ICDM*, pages 845–850, 2010.

[83] S. Günnemann, I. Färber, M. Rüdiger, and T. Seidl. Smvc: semi-supervised multi-view clustering in subspace projections. In *SIGKDD*, pages 253–262. ACM, 2014.

[84] S. Günnemann, I. Färber, and T. Seidl. Multi-view clustering using mixture models in subspace projections. In *SIGKDD*, pages 132–140. ACM, 2012.

[85] S. Günnemann, E. Müller, I. Färber, and T. Seidl. Detection of orthogonal concepts in subspaces of high dimensional data. CIKM '09, pages 1317–1326, New York, NY, USA, 2009. ACM.

[86] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[87] J. Han, Y. Sun, X. Yan, and P. S. Yu. Mining heterogeneous information networks. In *Tutorial at the 2010 ACM SIGKDD Conf. on Knowledge Discovery and Data Mining (KDD'10), Washington, DC*, 2010.

[88] M. H. Hansen and B. Yu. Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, 96(454):746–774, 2001.

[89] R. A. Harshman. Foundations of the parafac procedure: Models and conditions for an" explanatory" multi-modal factor analysis. 1970.

[90] X. He, J. Feng, B. Konte, S. T. Mai, and C. Plant. Relevant overlapping subspace clusters on categorical data. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 213–222, New York, NY, USA, 2014. ACM.

[91] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The MADlib Analytics Library: Or MAD Skills, the SQL. *Proc. VLDB Endow.*, 5(12):1700–1711, 2012.

[92] T. Hey, S. Tansley, and K. M. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery.* Microsoft Research, 2009.

[93] L. B. Holder, D. J. Cook, and S. Djoko. Substucture discovery in the subdue system. In *KDD Workshop*, pages 169–180, 1994.

[94] J. Hu, Q. Qian, J. Pei, R. Jin, and S. Zhu. Finding multiple stable clusterings. In *ICDM*, pages 171–180. IEEE, 2015.

[95] Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Min. Knowl. Discov.*, 2(3):283–304, 1998.

[96] L. Hubert. *Assignment methods in combinatorial data analysis.* M. Dekker, New York, N.Y., 1986.

[97] N. Hubig, J. Feng, X. He, A. Züfle, C. Plant, and C. Böhm. Detection of overlapping communities in attributed graphs. under review.

[98] N. Hubig, P. Fengler, A. Züfle, and S. Günnemann. Mining natural hazards: Unsupervised and supervised mining of large global spatio-temporal data. under review.

[99] N. Hubig and C. Plant. Information-theoretic non-redundant subspace clustering. In *PAKDD*, 2017 to appear.

[100] F. Hueske, M. Peters, M. J. Sax, A. Rheinländer, R. Bergmann, A. Krettek, and K. Tzoumas. Opening the Black Boxes in Data Flow Optimization. *Proc. VLDB Endow.*, 5(11):1256–1267, 2012.

[101] A. Hyv et al. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999.

[102] A. Hyvärinen and P. Hoyer. Emergence of phase-and shift-invariant features by decomposition of natural images into independent feature subspaces. *Neural computation*, 12(7):1705–1720, 2000.

[103] A. Hyvärinen and U. Köster. Fastisa: A fast fixed-point algorithm for independent subspace analysis. In *ESANN*, pages 371–376, 2006.

[104] Y. Ioannidis, R. Ramakrishnan, and L. Winger. Transitive closure algorithms based on graph traversal. *ACM Trans. Database Syst.*, 18(3):512–576, 1993.

[105] I. Jeon, E. E. Papalexakis, U. Kang, and C. Faloutsos. Haten2: Billion-scale tensor decompositions. In *ICDE*, 2015.

[106] A. Jindal, S. Madden, M. Castellanos, and M. Hsu. Graph analytics using vertica relational database. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 1191–1200, 2015.

[107] K. Kailing, H.-P. Kriegel, and P. Kröger. Density-connected subspace clustering for high-dimensional data. pages 246–257, 2004.

[108] K. Kanatani. Motion segmentation by subspace separation and model selection. In *IEEE ICCV*, volume 2, pages 586–591, 2001.

[109] U. Kang and C. Faloutsos. Beyond 'caveman communities': Hubs and spokes for graph compression and mining. In *ICDM*, pages 300–309, 2011.

[110] R. Kannan and S. Vempala. Spectral algorithms. *Found. Trends Theor. Comput. Sci.*, 4(3&#8211;4):157–288, Mar. 2009.

[111] J. Kaplan and M. DeMaria. Large-scale characteristics of rapidly intensifying tropical cyclones in the north atlantic basin. *Weather and forecasting*, 18(6):1093–1108, 2003.

[112] J. Kaplan et al. Evaluating environmental impacts on tropical cyclone rapid intensification predictability utilizing statistical models. *Weather and Forecasting*, 30(5):1374–1396, 2015.

[113] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *DAC*, pages 343–348, 1999.

[114] L. Katz. A new status index derived from sociometric analysis. *PSYCHOMETRIKA*, 18(1):39–43, 1953.

[115] A. Kemper and T. Neumann. HyPer: A Hybrid OLTP & OLAP Main Memory Database System Based on Virtual Memory Snapshots. In *ICDE 2011*, pages 195–206, 2011.

[116] A. Kemper and T. Neumann. Main-memory database systems. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, page 1310, 2014.

[117] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. pages 392–403, 1998.

[118] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison-Wesley, Reading, MA, 1993.

[119] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM.*, pages 455–500, 2009.

[120] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *TKDD*, 3(1):1, 2009.

[121] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. 3(1):1–58, 2009.

[122] H.-P. Kriegel, P. Kröger, and A. Zimek. Outlier detection techniques, 2010.

[123] P. Kroonenberg and J. ten Berge. The equivalence of tucker3 and parafac models with two components. *Chemometrics and Intelligent Laboratory Systems*, 106(1):21–26, 3 2011.

[124] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3), 2009.

[125] V. Leis, P. A. Boncz, A. Kemper, and T. Neumann. Morsel-driven parallelism: a numa-aware query evaluation framework for the many-core age. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 743–754, 2014.

[126] D. D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*, pages 4–15. Springer, 1998.

[127] D. Liben-Nowell and J. M. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559, 2003.

[128] B. Liebl, U. Nennstiel-Ratzel, R. von Kries, R. Fingerhut, B. Olgemöller, A. Zapf, and A. A. Roscher. Very high compliance in an expanded ms-ms-based newborn screening program despite written parental consent. *Preventive medicine*, 34(2):127–131, 2002.

[129] Y. Lim, U. Kang, and C. Faloutsos. Slashburn: Graph compression and mining beyond caveman communities. *IEEE Trans. Knowl. Data Eng.*, 26(12):3077–3089, 2014.

[130] S. Liu, S. Wang, F. Zhu, J. Zhang, and R. Krishnan. Hydra: Large-scale social identity linkage via heterogeneous behavior modeling. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 51–62. ACM, 2014.

[131] S. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Th.*, 28(2):129–137, 1982.

[132] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Graphlab: A new framework for parallel machine learning. *CoRR*, 2014.

[133] B. Ludäscher and N. Mamoulis, editors. *Scientific and Statistical Database Management, 20th International Conference, SSDBM 2008*, LNCS. Springer, 2008.

[134] R. Manmatha, T. Rath, and F. Feng. Modeling score distributions for combining the outputs of search engines. In *SIGIR*, pages 267–275. ACM, 2001.

[135] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers.

[136] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *NIPS*, pages 548–556, 2012.

[137] M. Meila. Comparing clusterings: an axiomatic view. In *ICML*, pages 577–584, 2005.

[138] R. S. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 11, pages 331–364. Tioga, 1983.

[139] G. Moise and J. Sander. Finding non-redundant, statistically significant regions in high dimensional data: a novel approach to projected and subspace clustering. In *KDD Conference*, pages 533–541, 2008.

[140] T. K. Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[141] F. Moser, R. Colak, A. Rafiey, and M. Ester. Mining cohesive patterns from graphs with feature vectors. In *SDM*, pages 593–604, 2009.

[142] W. E. Moustafa, A. Kimmig, A. Deshpande, and L. Getoor. Subgraph pattern matching over uncertain graphs with identity linkage uncertainty. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 904–915. IEEE, 2014.

[143] T. Mühlbauer, W. Rödiger, R. Seilbeck, A. Reiser, A. Kemper, and T. Neumann. Instant Loading for Main Memory Databases. *Proc. VLDB Endow.*, 6(14):1702–1713, 2013.

[144] E. Müller, I. Assent, S. Günnemann, R. Krieger, and T. Seidl. Relevant subspace clustering: Mining the most interesting non-redundant concepts in high dimensional data. In *ICDM*, pages 377–386, 2009.

[145] E. Müller, S. Günnemann, I. Assent, and T. Seidl. Evaluating clustering in subspace projections of high dimensional data. *PVLDB*, 2(1):1270–1281, 2009.

[146] E. Müller, F. Keller, S. Blanc, and K. Böhm. Outrules: A framework for outlier descriptions in multiple context spaces. In *ECML/PKDD (2)*, pages 828–832, 2012.

[147] H. Nagesh, S. Goil, and A. Choudhary. Adaptive grids for clustering massive data sets. In *Proceedings of the 1st SIAM ICDM, Chicago, IL*, volume 477. SIAM, 2001.

[148] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *Proceedings of the VLDB Endowment*, 4(9):539–550, 2011.

[149] T. Neumann. Efficiently Compiling Efficient Query Plans for Modern Hardware. *Proc. VLDB Endow.*, 4(9):539–550, 2011.

[150] T. Neumann, T. Mühlbauer, and A. Kemper. Fast serializable multi-version concurrency control for main-memory database systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 677–689, 2015.

[151] M. E. J. Newman. Clustering and preferential attachment in growing networks. *PHYS.REV.E*, 64:025102, 2001.

[152] M. E. J. Newman. *Phys. Rev. E74,036104*, 2006.

[153] D. Niu, J. G. Dy, and M. I. Jordan. Multiple non-redundant spectral clustering views. In J. Fürnkranz and T. Joachims, editors, *ICML*, pages 831–838, 2010.

[154] G. K. Orman, V. Labatut, and H. Cherifi. An empirical study of the relation between community structure and transitivity. *CoRR*, abs/1207.3234, 2012.

[155] V. Pandey, A. Kipf, D. Vorona, T. Mühlbauer, T. Neumann, and A. Kemper. High-performance geospatial analytics in hyperspace. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 2145–2148, 2016.

[156] E. Papalexakis, L. Akoglu, and D. Ience. Do more views of a graph help? community detection and clustering in multi-graphs. In *FUSION 2013*, pages 899–905, 2013.

[157] E. Papalexakis, N. Sidiropoulos, and R. Bro. From k-means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *Signal Processing, IEEE*, 61(2):493–506, 2013.

[158] E. E. Papalexakis, C. Faloutsos, and N. D. Sidiropoulos. Parcube: sparse parallelizable tensor decompositions. In *PKDD*, pages 521–536. Springer, 2012.

[159] L. Passing, M. Then, N. Hubig, H. Lang, M. Schreier, S. Günnemann, A. Kemper, and T. Neumann. Sql- and operator-centric data analytics in relational main-memory databases. In *EDBT*, 2017.

[160] J. M. Patel and D. J. DeWitt. Partition based spatial-merge join. In *ACM SIGMOD Record*, volume 25, pages 259–270. ACM, 1996.

[161] A. H. Phan and A. Cichocki. {PARAFAC} algorithms for large-scale problems. *Neurocomputing*, pages 1970 – 1984, 2011.

[162] C. Plant. Dependency clustering across measurement scales. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 361–369, New York, NY, USA, 2012. ACM.

[163] C. Plant, N. Hubig, S. Maurus, and A. Tonch. Seminar in current topics in information-theoretic data mining. *TUM Seminar*, 2014.

[164] F. Provost and T. Fawcett. Data science and its relationship to big data and data-driven decision making. *Big Data*, 1(1):51–59, 2013.

[165] T. Rakthanmanon, E. J. Keogh, S. Lonardi, and S. Evans. Mdl-based time series clustering. *Knowl. Inf. Syst.*, 33(2):371–399, 2012.

[166] M. Rienecker, M. Suarez, R. Todling, J. Bacmeister, L. Takacs, H. Liu, W. Gu, M. Sienkiewicz, R. Koster, R. Gelaro, et al. The geos-5 data assimilation system–documentation of versions 5.0. 1, 5.1. 0, and 5.2. 0. *NASA Tech. Memo*, 104606(27):2008, 2008.

[167] M. M. Rienecker, M. J. Suarez, R. Gelaro, R. Todling, J. Bacmeister, E. Liu, M. G. Bosilovich, S. D. Schubert, L. Takacs, G.-K. Kim, et al. MERRA: NASA's modern-era retrospective analysis for research and applications. *Journal of Climate*, 24(14):3624–3648, 2011.

[168] J. Rissanen. Modeling by shortest data description. *Automatica*, (14):465–471, 1978.

[169] J. Rissanen. An introduction to the mdl principle. Technical report, Helsinkin Institute for Information Technology, 2005.

[170] J. Rissanen. *Information and Complexity in Statistical Modeling*. Springer, 2007.

[171] S. Réjichi and F. Chaabane. Feature extraction using pca for vhr satellite image time series spatio-temporal classification. In *IGARSS*, pages 485–488, 2015.

[172] SAP SE. *SAP HANA Predictive Analysis Library (PAL) Reference, SAP HANA Platform SPS 09*, 2014.

[173] T. Schank and D. Wagner. Approximating clustering coefficient and transitivity. *J. Graph Algorithms Appl.*, 9(2):265–275, 2005.

[174] K. Sequeira and M. Zaki. Schism: A new approach for interesting subspace mining. In *ICDM*, pages 186–193. IEEE, 2004.

[175] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. *Journal of Machine Learning Research - Proceedings Track*, 5:488–495, 2009.

[176] J. Shi and J. Malik. Normalized cuts and image segmentation. In *CVPR*, pages 731–737, 1997.

[177] A. Silva, W. M. Jr., and M. J. Zaki. Mining attribute-structure correlated patterns in large attributed graphs. *PVLDB*, 5(5):466–477, 2012.

[178] M. Steinbach, P.-N. Tan, V. Kumar, S. Klooster, and C. Potter. Discovery of climate indices using clustering. In *KDD*, pages 446–455, 2003.

[179] Y. Sun, C. C. Aggarwal, and J. Han. Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. *PVLDB*, 5(5):394–405, 2012.

[180] Z. Szabó, B. Póczos, and A. Lőrincz. Separation theorem for independent subspace analysis and its consequences. *Pattern Recognition*, 45(4):1782–1791, 2012.

[181] P. Tamayo, C. Berger, M. Campos, J. Yarmus, B. Milenova, A. Mozes, M. Taft, M. Hornick, R. Krishnan, S. Thomas, M. Kelly, D. Mukhin, B. Haberstroh, S. Stephens, and J. Myczkowski. Oracle Data Mining. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 1315–1329. Springer US, 2005.

[182] M. Then, M. Kaufmann, A. Kemper, and T. Neumann. Evaluation of parallel graph loading techniques. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, GRADES '16, pages 4:1–4:6, New York, NY, USA, 2016. ACM.

[183] M. Then, L. Passing, N. Hubig, S. Günnemann, A. Kemper, and T. Neumann. Effiziente Integration von Data-und Graph-Mining-Algorithmen in relationale Datenbanksysteme. In *Proceedings of the LWA 2015 Workshops*, pages 45–49. CEUR-WS.org, 2015.

[184] W. R. Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, pages 234–240, 1970.

[185] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.

[186] A. K. Tung, X. Xu, and B. C. Ooi. CURLER: Finding and visualizing nonlinear correlation clusters. In *SIGMOD*, pages 467–478, 2005.

[187] S. van Dongen. *Graph Clustering by Flow Simulation.* Dissertation, University of Utrecht, 2000.

[188] N. X. Vinh and J. Epps. mincentropy: a novel information theoretic approach for the generation of alternative clusterings. In *ICDM*, pages 521–530. IEEE, 2010.

[189] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *ICML*, pages 1073–1080, New York, NY, USA, 2009. ACM.

[190] D. Wang and W. Ding. A hierarchical pattern learning framework for forecasting extreme weather events. *ICDM 2015*.

[191] R. L. Wears. Advanced statistics: statistical methods for analyzing cluster and cluster-randomized data. *Academic emergency medicine*, 9(4):330–341, 2002.

[192] J. J. Whang, D. F. Gleich, and I. S. Dhillon. Overlapping community detection using seed set expansion. CIKM ’13, pages 2099–2108.

[193] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 Algorithms in Data Mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[194] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Comput. Surv.*, 45(4):43:1–43:35, 2013.

[195] R. Xu, D. Wunsch, et al. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.

[196] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng. A model-based approach to attributed graph clustering. In *SIGMOD*, pages 505–516, 2012.

[197] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002.

[198] A. Y. Yang, J. Wright, Y. Ma, and S. S. Sastry. Unsupervised segmentation of natural images via lossy data compression. *Comput. Vis. Image Underst.*, 110(2):212–225, May 2008.

[199] J. Yang and J. Leskovec. Overlapping community detection at scale: A nonnegative matrix factorization approach. pages 587–596. ACM, 2013.

[200] R. Yang. A systematic classification investigation of rapid intensification of atlantic tropical cyclones with the ships database. *Weather and Forecasting*, 31(2):495–513, 2016.

[201] R. Yang, J. Tang, and M. Kafatos. Mining "optimal" conditions for rapid intensifications of tropical cyclones. *Program of the 19th Conference on Probability and Statistics, American Meteorological Society*, 2008.

[202] R. Yang, J. Tang, and D. Sun. Association rule data mining applications for atlantic tropical cyclone intensity changes. *Weather and Forecasting*, 26(3):337–353, 2011.

[203] W. Ye, S. Maurus, N. Hubig, and C. Plant. Generalized independent subspace clustering. In *ICDM*, 2016.

[204] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

[205] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. USENIX Association, 2010.

[206] A. Zhang, N. Fawaz, S. Ioannidis, and A. Montanari. Guess who rated this movie: Identifying users through subspace clustering. *CoRR*, abs/1208.1544, 2012.

[207] S. Zhang, H.-S. Wong, and Y. Shen. Generalized adjusted rand indices for cluster ensembles. *Pattern Recognition*, 45(6):2214 – 2226, 2012.

[208] Y. Zhang and D.-Y. Yeung. Overlapping community detection via bounded nonnegative matrix tri-factorization. In *KDD*, pages 606–614, 2012.

[209] H.-T. Zheng, H. Chen, Y. Jiang, S.-T. Xia, and H. Li. A two-step non-redundant subspace clustering approach. In *Semantic Web and Web Science*, Springer Proceedings in Complexity, pages 201–214. Springer New York, 2013.

[210] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.

[211] Y. Zhou, H. Cheng, and J. X. Yu. Clustering large attributed graphs: An efficient incremental approach. In *ICDM*, pages 689–698, 2010.