A Flexible Framework to Monitor Evolution of Clusters

by

Victor do Nascimento Silva

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Nowadays, the volume of collected data and the size of datasets raise various challenges in the field of data mining. One of such challenges is to, given a dataset, monitor a set of data points and its changes over a period of time. Previously, this monitoring has been done using pattern matching, spatial and velocity data profiling, and transition extraction, among others. Variations of this problem are found in many fields, such as social network analysis, social sciences, finance, environmental sciences, and epidemics. Monitoring sets of data can give insights into how a subset of data behaves and help explaining what happened to that set over a period of time. For instance, in behavioral sciences, it might explain how groups of individuals behave. Another example is that in finance, insights might help analysts to trace a set of companies and their behaviors in a portfolio, which can help them in their decision-making process.

In this thesis we address the problem of cluster monitoring over time. We tackle this problem by proposing a pattern-based framework. The state-of-the-art addresses the problem of cluster monitoring by finding transitions to describe changes in the data over time. Our approach instead detects much more complex patterns by mining for patterns instead of using a set of rules to detect a set of transitions.

While transitions are an effective way to describe change, they might be restrictive when describing complex behaviors. To tackle this problem, our pattern-based framework monitors the evolution of a cluster using an evolution graph rather than transitions. This graph can then be mined using a subgraph detection algorithm to find useful patterns. One of the advantages of a pattern-based framework over a transition-based framework is the capacity of detecting more complex behaviors of data. We evaluate our approaches using two real-world datasets. Experiments show that our pattern-based framework detects both simple and complex transitions. We also show that our proposed framework detects patterns that would not be detected otherwise.

To my siblings,

Raissa, Emanuella, João and Natã,

my children, by heart, that kept me dreaming.

The increase of disorder or entropy with time is one example of what is called an arrow of time, something that distinguishes the past from the future, giving a direction to time.

– Stephen Hawking, A brief history of time, 1988.

Acknowledgements

Firstly, I would like to thank my supervisor, Dr. Jörg Sander. Your calm personality and analytical approach helped me to succeed. I am grateful for your mentorship and for guiding me into uncharted waters. I am both blessed and lucky to have had you as supervisor through my studies. Thank you for always sharing your knowledge, your patience, your support and your passion for research.

To my committee, thank you for your valuable feedback. Your criticism allowed me to think further about the subject. Your continuous support allowed me to improve it.

Thank you to Marlos and Poliana, those who encouraged me to come to Canada and pursue my studies at the University of Alberta. I will be forever in debt with you for all your help.

My lab colleagues and that came along the way: Megha, Antonio(s), Geörg and many others. Thank you for insightful conversations and for listening to me talk about my research. You made the journey more fun.

To FGSR's Associate Deans, thank you for your support. I would like to extend this acknowledgment to FGSR as a whole, you do a fantastic job in supporting studies through their career at this University.

Thank you to the Department of Computing Science, the Chair and all the associates in the main office for helping me with all the challenges and bureaucracy. You made my life easier always holding a smile on your face.

To my friends and family, which I left behind physically, but that are always in my mind and in my heart, thank you. In special to Dayane, for keeping me close and pushing me forward. Thank you to Lena, your library was the perfect place to recharge and get back to writing. Your love for people and for doing good to the world truly changed the way I see life.

Finally, during these three years my life has been like an airport. Many came, many are gone, some stayed, but many of them are gone forever. For those who had the opportunity to land, for those who had layovers, for those who stayed, and those who are gone: thank you. In some way you are part of this thesis, because you have affected my life intensely and these encounters left their marks engraved in this text. In the end, *"Airports have seen more sincere kisses than wedding halls"*.

Contents

1 Introduction **1**
1.1 Contributions 3
1.2 Dissertation Layout 3

2 Background and Related Work **4**
2.1 Clustering 4
2.1.1 Partitional Clustering 4
2.1.2 Hierarchical Clustering 5
2.2 Density-Based Clustering 6
2.3 Spectral Clustering 7
2.4 Graph-based Clustering 8
2.5 Cluster Monitoring 9
2.5.1 Change Detection 9
2.5.2 Transition-based Change Detection 10
2.5.3 Cluster Evolution Summarization 18
2.5.4 Community Mining 19

3 Methodology **23**
3.1 Motivation 23
3.2 A pattern-based framework for cluster monitoring 29
3.2.1 Segmentation 30
3.2.2 Clustering 33
3.2.3 Similarity 34
3.2.4 Evolution Graph 36
3.2.5 Tracking Sets 37
3.2.6 Pattern Discovery 39
3.2.7 Discussion 43
3.2.8 Summary 44
3.3 A transition-based framework for cluster monitoring 44
3.3.1 Similarity and transition extraction 44
3.3.2 Discussion 48
3.4 Final remarks 48

4 Experiments **50**
4.1 NSERC Dataset experiments 50
4.1.1 Dataset 51
4.1.2 Experiment Overview 52
4.1.3 Data Processing 53
4.1.4 Document representation 54
4.1.5 Segmentation 55
4.1.6 Clustering 55
4.1.7 Similarity 58
4.1.8 Evolution Graph 58

4.1.9 Experiment 1 - Frequent Topic 59
4.1.10 Experiment 2 - Investigating Topics and Funding . . . 61
4.1.11 Experiment 3 - Transition-based cluster monitoring . . 63
4.2 S&P 500 Dataset 67
4.2.1 Data Processing 67
4.2.2 Stock Representation 68
4.2.3 Segmentation 69
4.2.4 Clustering 69
4.2.5 Similarity 70
4.2.6 Evolution Graph and Visualization 72
4.2.7 Experiment 1 - Transition-based Pattern Detection . . 72
4.2.8 Experiment 2 - Transition-free Pattern Detection . . . 75
4.3 Discussion 77

5 Conclusion **78**
5.1 Future Work 78

References **80**

List of Tables

2.1 Cluster transitions in MEC 16

4.1 Transitions detected by MONIC when varying λ 64
4.2 Transitions detected by MONIC when varying τ 64
4.3 Transitions detected by MONIC when varying λ 73
4.4 Transitions detected by MONIC when varying τ 73

List of Figures

2.1 An example of k-means, circles represent data points and squares represent centroids. (a) The centroids initial placement. (b) Result after the algorithm has converged. 5
2.2 Iris dataset plotted using the first two attributes of the dataset (left) and a dendrogram created from that dataset by the HAC algorithm using Euclidean distance and Ward's method (right). 7
2.3 Example of clustering algorithms applied to a synthetic dataset. (a) k-means with $k = 2$; (b) spectral clustering with $k = 3$; (c) DBSCAN. 8
2.4 The bipartite graph representation proposed by MEC. 18

3.1 Diagram with clusters and its respective similarity values between two timestamps. 24
3.2 Diagram with clusters and its respective similarity values between three timestamps. 26
3.3 Labeled Diagram derived from figures 3.1 (a) and 3.2 (b) . . . 27
3.4 Patterns with four or more nodes and that span at least three segments found in the diagram in Figure 3.3 (b) 28
3.5 An illustrated temporal segment and its elements. In green the t_{start}, in blue the time window size Δ and in orange the t_{end}. . 32
3.6 Example of a segmentation of a temporal dataset TD. 32
3.7 Example of an EG, nodes that precede the tracked set Q point to Q, and Q points to the sets that succeed it. 38
3.8 Two isomorphic graphs, where (a) is the initial EG that is labeled into (b). 41
3.9 Diagram showing the pipeline of the pattern-based framework. 44
3.10 A toy evolution graph illustrating transitions 46
3.11 Example of count change transition. Size indicates the cardinality or a cluster, the larger the node the higher the cardinality. On top a $count_+$ transition, in the bottom a $density_-$ transition. 47
3.12 Example of density change transition. Color indicates the density, the darker the denser. On top a $density_+$ transition, in the middle a $density_-$ transition and, in the bottom an extreme $density_-$ going from dense to outlier group (represented by a square). 48

4.1 A word cloud w.r.t. the tf-idf representation of the whole data. 55
4.2 percentage of outliers w.r.t. `min_cluster_size`(a), and number of cluster w.r.t `min_cluster_size`(b) 57
4.3 percentage of outliers w.r.t. `min_samples`(a), and number of cluster w.r.t `min_samples`(b) 57
4.4 Clustering of the dataset using HDBSCAN* with `min_cluster_size = 2` and `min_samples = 5`. Data projected using TSNE. . . . 58

4.5 Monitoring of the terms “wireless” and “network” from 2006 to 2009 (a); and, from 2015 to 2017 (b). The edges that show evolution of the topic are coloured in red. 61
4.6 Number of funded proposals per year (a) and Amount of funding awarded per year, in millions of dollars (b) 63
4.7 Average cosine similarity value and standard deviation. 64
4.8 In both diagrams a cluster had high similarity value to two other clusters in the subsequent year. The pattern is detected even though the topics are different. 66
4.9 A stock was represented by its vector of prices given a segment 68
4.10 Cluster contents in the S&P500 dataset 70
4.11 Cluster contents in the S&P500 dataset 71
4.12 Most frequent transition-like pattern detected by our framework when using Cosine Similarity 75
4.13 Most frequent transition-like pattern detected by our framework when using Jaccard Similarity 76
4.14 Most frequent pattern when $\epsilon = 3$ 76

1 Evolution Graph of the NSERC Dataset from 2013 to 2017 and edges with similarity values of at least 0.5. A black word cloud represents an outlier group. 85

Chapter 1

Introduction

In the last decades we find that the amount of data and the speed in which it flows is growing faster than ever. Millions of queries and transactions are processed every second by search engines, online marketplaces, and other services. With the paradigm of data streams, the interest in tracking the flow of data and its changes over time has caught the interest of scientists. This interest started with the study of data streams and the paradigm of *change mining*. An early work in the field points out that the mining of change in data streams is one of the core problems in data mining [15]. The problem of change mining was later tackled by many researchers, and one of the main features deemed valuable in this problem by many scientists is *time* [9]. The traditional methods of data mining consider the data to be static, which is not the case in many scenarios. Data is changing, and new information is continuously incoming, which requires new techniques for the analysis of data.

The field of change mining studies the evolving nature of the data over time. The goal of change mining is to discover the changes that happen in the data and extract information from these changes. The detection of change in data can help in the adaptation of learned models, such as clusters, in dynamic datasets. Moreover, change mining could help detecting important features in real-world datasets, e.g. fake transactions, network intrusion, social influence, topics in textual datasets, and migration patterns in a population. Despite its general importance, the term *change mining* became less popular in this decade, being substituted by terms that address specific sub-fields, e.g. outlier detection, novelty detection, concept drift, and cluster monitoring.

Cluster monitoring is a sub-field of change mining that is concerned with tracing clusters over time, identifying transitions and extracting insights from the discovered transitions. These goals are shared with other sub-fields of change mining, e.g. complex networks and community mining. Moreover, monitoring clusters over time might help in the knowledge discovery process, e.g. the discovery of different groups involved in a data stream, observing when groups form, come together and fall apart. These observations are important for fields like Finance, Marketing and Social Networks. For instance, in Finance it is possible to monitor a portfolio and market sectors.

A closely related field to cluster monitoring is community mining. This field is concerned with describing networks, especially social networks, its events and evolution over time. Differently from cluster monitoring, datasets in community mining are graphs, where nodes represent individuals and edges represent some kind of relationship between those individuals. This field became very popular in the last few decades, but it tackles a problem that is somewhat different from cluster monitoring. While cluster monitoring addresses the problem of tracking clusters, community mining addresses the problem of discovering and tracking communities. Communities are groups of data points that have a relationship among them, e.g., in social network a friendship is a relationship. Data in community mining is commonly represented by a graph, while in cluster monitoring it is represented by a set of data points.

This thesis proposes a flexible framework for cluster tracking. We tackle the problem of cluster monitoring where our goals are: a) to discover frequent-patterns of arbitrary size; and b) to monitor a cluster evolution over time.

To tackle this problem, we propose a pattern-based framework for monitoring clusters over time. In this framework we can use different functions to compute similarity between nodes. Different parameters can be adapted in order to improve the knowledge discovery process w.r.t. a dataset.

1.1 Contributions

The main contributions of this thesis are:

- a pattern-based framework, that detects patterns of arbitrary size (Section 3.2);
- an extension the current transition-base framework for cluster monitoring (Section 3.3);

1.2 Dissertation Layout

This thesis contains five chapters organized as follows: Chapter 2 introduces the necessary background concepts to understand this thesis as well as the related work. In Chapter 3, I discuss the three proposed frameworks for cluster monitoring. Chapter 4 contains experiments that were performed to evaluate the proposed frameworks. Finally, in Chapter 5, I present the conclusions and avenues for future work.

Chapter 2

Background and Related Work

2.1 Clustering

Clustering is the process of finding groups in a dataset such that elements in the same group are more similar to each other than to elements in the other groups. In this section I briefly discuss clustering techniques that are relevant for this dissertation, namely: partitional clustering, hierarchical clustering, density-based clustering, spectral clustering and graph clustering.

2.1.1 Partitional Clustering

The goal of partitional clustering is to divide a set of data points into k disjoint sets (clusters) while minimizing an objective function. Partitional clustering can be parametric or non-parametric. Parametric means that the clustering algorithm must fit a distribution, e.g., EM clustering, while non-parametric means that the clustering algorithm does not have to fit a distribution, e.g., mean-shift. In this thesis I only use parametric partitional clustering, which I discuss further below.

2.1.1.1 K-means

The most well-known partitional clustering algorithm is k-Means [22]. K-means divides the space into k partitions, where k is a user-provided parameter. A pseudo-code of k-means is presented in Algorithm 1.

Algorithm 1 K-means

```
1: procedure K-MEANS(D, k)
2:     Place $k$ centroids in the clustering space using a predefined criterion;
3:     Assign each data point $d \in D$ to its nearest centroid;
4:     Update the centroid to the mean of its attributed data points;
5:     *repeat* 3-4 until a stopping criterion is met.
```

There are several criteria that can be used to initially place the centroids in step 1, e.g. random, or grid. The most common stopping criterion for k-means are: i) when a number of iterations is reached; ii) when data points do not change clusters after update. We illustrate k-Means in Figure 2.1.

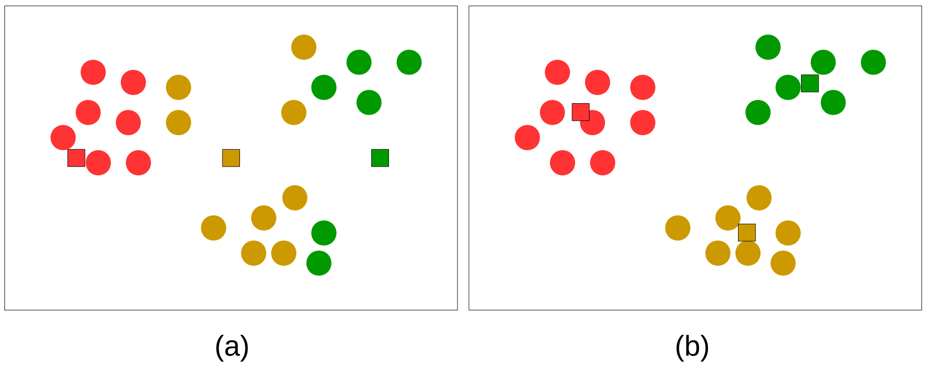


Figure 2.1: An example of k-means, circles represent data points and squares represent centroids. (a) The centroids initial placement. (b) Result after the algorithm has converged.

2.1.2 Hierarchical Clustering

Hierarchical clustering is a technique that seeks to establish a hierarchical relationship between data points and clusters. There are two common approaches to hierarchical clustering: agglomerative, also called bottom-up; and, divisive, also called top-down.

In agglomerative clustering, each point in the dataset is initially considered a singleton cluster. Each cluster is then merged to another following a linkage criterion. A linkage criterion is a function that computes the distance between two sets of datapoints as a function of the pairwise distance between observations in each set. The merging process is repeated until a single cluster that contains all data points in the dataset is created [53]. To obtain flat clusters from a hierarchy, it is necessary to choose a cut threshold that will be used to select what data points will be part of each cluster based on the final dendrogram.

In divisive clustering, the whole dataset is initially a singleton super cluster, which is then divided into two smaller clusters given a division criterion. The new smaller clusters are then recursively divided until all clusters contain a single data point [31], [47].

One of the most common approaches among hierarchical clustering algorithms is *Hierarchical Agglomerative Clustering* (HAC). In HAC the clusters are initially represented by each point in the dataset. The distance between each pair of clusters is then computed and clusters are merged according to a linkage criterion. For example, when using Euclidean distance as the distance function and single linkage as the linkage criterion the algorithm computes the distance between clusters as the minimum distance between any two points in the clusters. Other examples of linkage criteria are complete linkage, average linkage, centroid linkage and Ward's method. I illustrate a dendrogram resulting from the execution of HAC with Ward's method as criterion to the Iris Dataset[1] [19] in Figure 2.2.

2.2 Density-Based Clustering

Density-based clustering is a non-parametric clustering approach that takes into account the density of the data, finding high-density regions separated by low-density regions in the clustering space to find clusters. Among the advantages of density-based methods is their ability to distinguish cluster points from outliers, as well as to find arbitrarily shaped clusters. The most popular

[1]The Iris dataset is composed of characteristics of three different species of flowers.

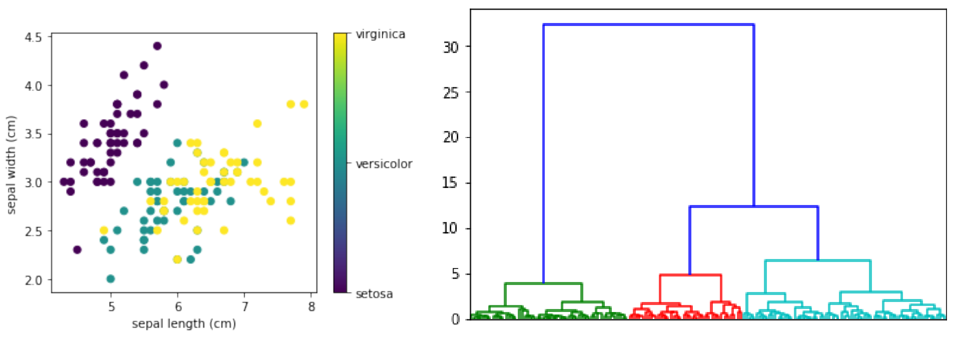


Figure 2.2: Iris dataset plotted using the first two attributes of the dataset (left) and a dendrogram created from that dataset by the HAC algorithm using Euclidean distance and Ward's method (right).

density-based clustering algorithm is DBSCAN [17], which uses the concept of density-reachability to find clusters. DBSCAN requires two parameters: `minPts`, which indicates the minimum number of points q that are within a distance ε of a point p for p to be considered a *core point*; and ε, which indicates the maximum distance from a point q to another core point q to be considered directly reachable. Points that do not have at least `minPts` points within distance ε but are reachable from a core point are called *border points* and are still part of a cluster. However, if a point is not a border point nor a core point, then it is labeled as an *outlier* or *noise*.

2.3 Spectral Clustering

Spectral clustering is a non-parametric clustering technique that can find clusters of arbitrary shape [35], [52]. This technique is commonly composed of three steps: i) create a similarity graph from the data; ii) compute the first k eigenvectors from the graph's Laplacian matrix; iii) run a clustering algorithm (normally k-means) to find k clusters. This approach is efficient when searching for a small number of clusters.

Spectral Clustering is most commonly used in the fields of community tracking and social network analysis. This is because in these fields the dataset is commonly represented by a graph where the edges are relationships between data points. Figure 2.3 illustrates the application of k-means, spectral clustering and DBSCAN on a synthetic dataset. The approaches present different results w.r.t. the presented dataset. The choice of a clustering algorithm is dependent on the type of data and the challenges to be addressed.

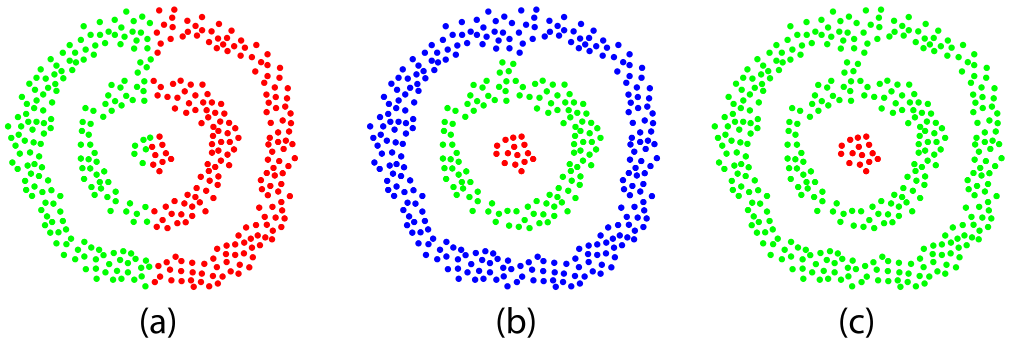


Figure 2.3: Example of clustering algorithms applied to a synthetic dataset. (a) k-means with $k = 2$; (b) spectral clustering with $k = 3$; (c) DBSCAN.

2.4 Graph-based Clustering

An influential clustering technique in the field of community mining and social networks, as well as biological sciences, is the Markov Cluster Algorithm (MCL) [10], [51], which is a graph clustering technique. MCL creates a matrix to represent the graph and then performs consecutive expansion and inflation of the matrix until convergence. The expansion process is controlled by a number e, where the matrix's *e-th* power is computed. This process controls the connection between different regions of the graph. The inflation process is controlled by a non-negative number r and consists in computing the r exponent of a column, followed by a normalization of the said column. The inflation process is responsible for strengthening strong connections and weakening weak connections in a graph. The parameter r controls the granularity of the clusters by adjusting how fast the weakening and strengthening will occur. The expansion and inflation processes are repeated until convergence. The result is a graph to be interpreted where the clusters are a set of connected components.

One drawback of this approach is that convergence is not guaranteed, although it occurs in most of the cases [51]. Moreover, this technique has problems in detecting clusters with large diameter, which might also increase convergence time.

2.5 Cluster Monitoring

Cluster monitoring is fundamentally about the detection of changes in datasets using clusters as change indicators. Researchers have long been curious to explain and extract insights from datasets that change over time [2]. Various paradigms and interpretations have been presented to this problem. In this section I cover topics related to the problem of cluster monitoring, namely: change detection, transition detection, cluster evolution, summarization of transitions, and community mining.

2.5.1 Change Detection

One of the simplest approaches to the problem of change detection is the FOCUS framework [20]. FOCUS uses statistical analysis to compare the difference between distinct datasets. The framework collects measures of deviation between models learned from the datasets and then use these measures to evaluate if two datasets are statistically different.

The PAM framework [7] proposes to monitor patterns, such as association rules, sequences and clusters, that might occur in data. The framework first mines the dataset in order to find frequent patterns and association rules. The mined patterns and rules are then modelled as data points with a temporal variable, called temporal objects. The mined patterns are then searched for in the data that arrives in the future. There is the assumption that the mined patterns are recurrent and occur in the future because they happened in the past. This might not be the true in cases where the patterns in a dataset are changing constantly or where patterns captured are noise.

The PANDA framework [8] computes the similarity value between patterns, such as association rules, cluster and keywords. PANDA defines a simple pattern as a pattern that cannot be decomposed, and a complex pattern as a pattern that is composed of multiple simple patterns and/or other complex patterns. The framework compares simple and complex patterns to find recursive characteristics in complex patterns.

2.5.2 Transition-based Change Detection

The concept of change and transitions in data is one of the central ideas in cluster monitoring. Several authors have discussed both change and transitions, proposing different methods to explore the change and nomenclatures for the transitions. In [11] the authors proposed an approach based on the summarization of a data stream using a HE-Tree (Hierarchical Entropy Tree). The work is concerned with detecting change in the structure of the data by finding the optimal number of clusters in subsequent snapshots, i.e. finding the best clustering at each snapshot. They then compare the number of clusters in subsequent snapshots, labeling a change when the number is different or when the change is larger than a threshold.

In [1], [2] the author proposes an analysis based on velocity and spatial profiles of the data, finding three types of transitions: *coagulation*, *dispersion* and *shift*. A *coagulation* occurs when the data moves in the direction of a single point in the space, making the cluster larger in terms of density and number of data points. The opposite of a coagulation is the *dispersion* that occurs when the data moves in the opposite direction of a point in the space, making the cluster less dense and smaller in terms of number of data points. Finally, when the dispersion and coagulation occur in adjacent areas, if there is data that moves from one central point of dispersion towards a central point of coagulation the data is said to be *shifting* from one point to another.

Further, in [56] the authors propose a framework that extracts frequent spatial object association patters (SOAP). To find SOAPs they propose two measures: support, that measures the number of snapshots that a given SOAP occurs in; and realization, that measures the smallest count of a given SOAP in all snapshots where it appears. The support measure is used to determine if a SOAP is frequent, given a user defined threshold. The realization measure is used to determine if a SOAP is prevalent in the dataset given another user defined threshold. These conditions are then used to find three different kinds of transitions: formation, when a SOAP count changes from zero to non-zero; dissipation, when the SOAP count changes to zero; and continuation, when at least one SOAP exists in two subsequent snapshots.

Falkowski et al. [18] propose a different view on transitions and how to extract them. They first use a graph dataset, using betweenness as a parameter for a hierarchical clustering algorithm. Then, four types of transitions can be detected: growing, declining, merging and splitting. These transitions are found by structurally comparing the clusters using several characteristics such as density, Euclidean distance, and correlation.

2.5.2.1 MONIC

The MONIC framework [48] is one of the main approaches in the field of cluster monitoring, building up on PAM and PANDA. Unlike its predecessors, it offers a more general idea on how to track cluster changes over time. The idea developed in MONIC is that a cluster exhibit simple state transitions that can be detected using different thresholds.

MONIC makes very few assumptions about a clustering, proposing only that the clusters are mutually exclusive and that the set of all clusters, covers the entire dataset. In the case of clustering algorithms that generate outliers, MONIC assumes a cleaning step is performed, where all outliers are removed from the dataset.

MONIC also defines an ageing function over the data. This function ensures that the data obtained at a timestep t_{i-1} is less important than data obtained at t_i. MONIC allows the ageing process to be performed by a decay function or by a sliding window. When using the decay function approach, a mathematical function will define the age of a given data point w.r.t its time t. In contrast, a sliding window method consists of defining a window of width w and attributing the same age value to all the data points within the defined window.

Given a clustering, MONIC monitors the change over time by comparing clusters in subsequent clusterings using a similarity function. Using these similarity values, MONIC extracts transitions based on a set of rules and thresholds. MONIC authors propose an *asymmetrical* similarity measure between clusters, measuring the best *match* (if any) of a cluster at timestep t_i when compared to all clusters at timestep t_{i+1}.

MONIC starts the analysis of the clustering over time by extracting the similarity between all pairs of clusters in subsequent clusterings and storing the computed similarity values in a matrix that is later used to extract the cluster transitions. Another concept in the MONIC framework is the *cluster match*. A cluster match is the event in with two subsequent clusters have similarity greater than a match threshold τ.

Because of how MONIC defines the match, two scenarios are possible: i) a cluster matches with only one cluster; or ii) a cluster matches with exactly two clusters. In MONIC, the match threshold is set in the interval $[0.5, 1]$, therefore, two clusters may equally overlap by 0.5 with a previous cluster. In this event MONIC states that a tie breaking method is applied.

Furthermore, MONIC defines external transitions using the similarity matrix and cluster matches. Before setting a cluster as a survival it verifies a secondary match using a threshold λ to check if that cluster is involved in a split transition. A cluster is said to have split if it matches with two or more clusters given a threshold λ and the sum of the similarity values of all matched clusters are at least τ.

To find transitions, MONIC first searches for the external transitions. External transitions are those that might change the clustering space itself. There are five different external transitions in MONIC: survival, split, absorption, disappearance, and emergence.

MONIC searches for external transitions between clusters iterating the snapshots created from a dataset. The process first searches for the existence of survival. A cluster is said to have *survived* if it matches one and only one cluster in the next clustering. If more than one cluster in the current clustering matches a cluster in the next clustering, each cluster that matches the same cluster in the next snapshot is said to have been *absorbed* by the cluster which they have matched. Survival and absorption are not possible at the same time, as survival is a one-to-one match and absorption is a many-to-one match. After searching for survival and absorption, the framework searches for splits. A threshold λ is established and the similarity value between the clusters in two subsequent clusterings are searched for mappings one-to-many. When one cluster in the current clusterings matches to multiple clusters in a subsequent clustering with similarity of at least λ, and the sum of all similarity values is at least τ, then its said that the matched cluster has *split* into the set of clusters it matched to in the next clustering. MONIC requires that $\lambda < \tau$ at all times to prevent degenerate cases. Finally, if a cluster in the current clustering does not have an external transition it is said to have *disappeared*. Likewise, a cluster in the current clustering that is not a result of a transition is said to have *emerged*.

MONIC also searches for *internal transitions*, in which a cluster changes its size, structure, or location. There are four possible internal transitions: size, compactness, location and no change.

A change in size, i.e. shrinking, or expanding, happens when the number of data points of the cluster changes. The compactness of a cluster is represented by the standard deviation. If the cluster's standard deviation changes by more than δ, it is said to have become more compact or more diffuse. For the location transition MONIC covers two cases: metric spaces and non-metric spaces. In a metric space the location transition indicates that a cluster has shifted, indicated by a change of its mean value by more than τ_1. On the other hand, if the cluster is represented in a non-metric space, then the skewness is used as the measure of location. A cluster is said to have changed its distribution if its skewness has changed by more than τ_2. The authors evaluate MONIC using the ACM library section H.2.8 dataset and shows that MONIC is capable of tracking clusters over time.

2.5.2.2 MONIC+

In a later paper, Ntoutsi et. al. [37] extended MONIC to exploit the characteristics of particular types of clusters in a framework named MONIC+. According to the authors, MONIC was incapable of exploiting the characteristics of specific cluster types that were generated by the different clustering methods proposed in the literature. In MONIC+ the authors explored three specific types of clusters: i) metric dataset-independent clusters; ii) non-metric set of data-dependent records; and iii) distribution clusters. These types of clusters are exploited in order to provide a more specific analysis, proposing different similarity functions for each of them. The authors also noted that not all transitions can be detected for all cluster types, having an in-depth discussion of type dependent transitions. Finally, they experimented with their approach in two different datasets: a synthetic dataset and the KDDCup 1999 Network Intrusion dataset [14]. The authors noted that evaluating a framework of cluster tracking is a challenge, and that there is no gold standard in the literature. The current evaluation method consists of running experiments in both synthetic and real-world data for empirical and qualitative evaluation.

2.5.2.3 MEC

An approach for monitoring cluster transitions similar to MONIC was presented in MClusT [39], where the framework uses a different similarity function based on conditional probabilities. Moreover, because the authors identified the need of a more straightforward way to visualize the results of the cluster tracking and insight extraction, they proposed a visualization tool using bipartite graphs. The MClusT framework builds upon MONIC, proposing a new taxonomy for the transitions that are found by cluster tracking and a new similarity function. First, MClusT separates the data into snapshots and then clusters each snapshot, finding the similarity between clusters from two subsequent snapshots using a similarity function. The result of this process is then visualized using a bipartite graph where the nodes are the clusters discovered by k-means or HAC. The edges of the bipartite graph indicate a connection between the generated nodes. Using a weighing function, MClusT weighs the edges of the bipartite graph based on the similarity between two given clusters. A pruning process is then applied to the bipartite graph where edges that have weight below a given threshold are removed. The framework is applied to a dataset from macroeconomics and two case studies: a) Portuguese activity sectors; and b) Portuguese regional development index.

MEC [41] is a framework for cluster monitoring that was built on the foundations of MONIC. In its early version, we find a more general approach which allows for enumerated data, which is uncompressed data where all the data points are preserved; and generalized data, which uses statistical methods, like the mean, to represent the dataset in compressed form [40]. However, because of data loss, the latest version of MEC uses the enumerated representation only.

The overall method is similar to MONIC: The data is collected, segmented, clustered and then analyzed. MEC extends MONIC in two ways: i) it uses a new weight function; and ii) it includes a visualization tool in the form of a bipartite graph [39]. Moreover, MEC proposes the standardization of the transition nomenclature found in eight previous different schemes [1], [2], [6], [11], [18], [48], [56]. We reproduce the proposed taxonomy in Table 2.1 and use it when we refer to transitions henceforth.

Table 2.1: Cluster transitions in MEC

Mathematical Notation	Description
$\emptyset \to C'_{t_j}$	Cluster's Birth
$C_{t_i} \to \emptyset$	Cluster's Death
$C_{t_i} \overset{\subsetneq}{\to} \{C'_{1,t_j}, ..., C'_{r,t_j}\}$	Split of a Cluster into r clusters
$\{C_{1,t_i}, ..., C_{p,t_i}\} \to C'_{t_j}$	Merge of p clusters into one cluster
$C_{t_i} \to C'_{t_j}$	Cluster's Survival

While MONIC authors prefer term *absorption* that represents a transition w.r.t. a single cluster in the current clustering, MEC authors prefer a set approach, adopting the term *merge* that represents a transition w.r.t. a subset of clusters in the current clustering. Another major difference is that MEC does not track internal transitions, focusing only on the changes that happen in the clustering space over time and not on the internal changes in each cluster composition.

MEC has the main goal of tracking clusters in labeled datasets. Because of this goal, MEC uses a conditional probability function which requires that the labels are consistent over all snapshots. This makes MEC less general than MONIC since it requires a more specific type of data. The MEC similarity value is computed as:

$$Sim(C, C') = P(X \in C' | X \in C) = \frac{\sum P(x \in C \cap C')}{\sum P(x \in C)} \tag{2.1}$$

This value is computed for pairs of clusters (C, C') where $C \in \xi$ and, $C' \in \xi'$, where ξ and ξ' are clusterings. These weights are the equivalent to the values created by the similarity function in MONIC. In contrast to MONIC, MEC does not compute a similarity matrix, storing the processed dataset in a bipartite graph. MEC maps each clustering to be a color in the bipartite graph, e.g. the current clustering would be a color and the subsequent clustering would be a distinct color. Further, each cluster in a given clustering is mapped into a node in the bipartite graph. Finally, given two subsequent clusterings, an edge is added between all clusters in the current clustering and all clusters in the subsequent clustering, weighed by the similarity function (Equation 2.1).

The computed bipartite graph (Figure 2.4), is used for visualization of the transitions extracted by the MEC framework. This tool allows for the data scientist to better understand the behavior of the clusters over time. The visualization also provides a more intuitive tool to visualize the data, aiding the process of knowledge discovery [41]. One of MEC's visualization tool drawbacks is that the bipartite graph model does not represent internal transitions, such as expansion and shrinking.

MEC is evaluated using two distinct datasets: a) Portuguese activity sectors, also discussed in the MClusT paper; and b) European companies' dataset. The evaluation consisted in running experiments using the framework in the process of knowledge discovery, tuning parameters and manually looking for

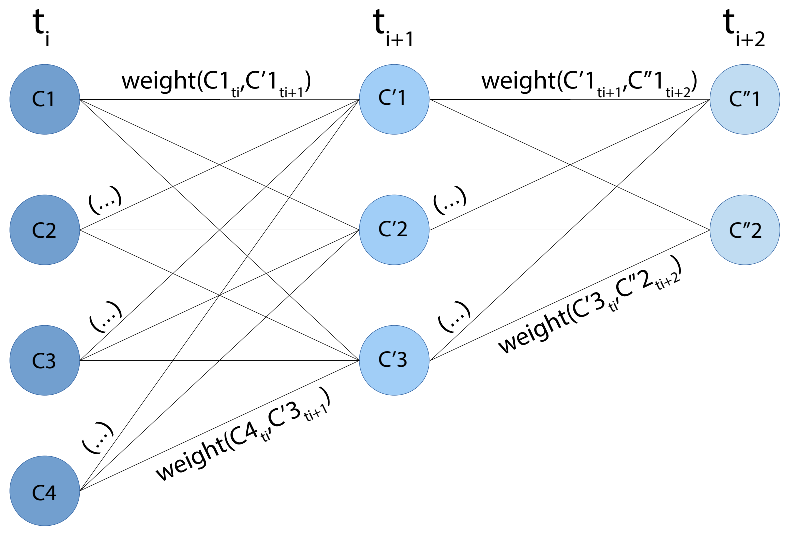


Figure 2.4: The bipartite graph representation proposed by MEC.

insights based on prior knowledge. The authors claim that no strong conclusions are drawn from the second dataset since specific knowledge from finance was needed.

2.5.3 Cluster Evolution Summarization

In FINGERPRINT [36], [38] the authors propose a model to summarize the evolution of clusters through the compression of external transitions found with the MONIC framework. FINGERPRINT uses two different approaches to summarize the transitions: batch summarization and incremental summarization. The batch summarization approach takes the entire dataset as a parameter. On the other hand, the incremental method can continually process a data stream. When comparing the aforementioned methods, the performance of the batch method is more robust to information loss and performs more data compression than the incremental method.

To summarize the cluster tracking, FINGERPRINT clusters the data stream, separating it into snapshots and applying k-means to each snapshot. Further, it transforms the clustered dataset into a structure named Evolution Graph

(EG) that is the equivalent of the bipartite graph introduced by MEC. Each node of an EG represents a cluster, and each edge between clusters in an EG represents an overlap between two clusters in adjacent snapshots. FINGERPRINT focuses only on the compression survival transition, and datasets where this transition does not occur often do not benefit from this technique.

2.5.4 Community Mining

Community mining is the field of study that investigates Communities, e.g., Social Networks, its properties and characteristics. The main difference between community mining and cluster monitoring is that in community mining the data is modeled as a graph, where the edges represent a relationship between data points. Moreover, community mining is also concerned with monitoring the behaviour of specific data points, while cluster monitoring addresses the changes in clusters. One of the open questions in community mining is "How to track a community over time?". The referred question is similar to the one I am trying to tackle in this dissertation; therefore I discuss research in community mining that is related to this work.

In [6] the authors identified events in a community evolution over time with the goal of discovering insights. Events are change indicators, equivalent to MONIC transitions. The communities are modelled as a graph, where each data point is a node and if there is a relationship between two given data points in a community there is an edge between them, e.g. in a social network a relationship would be a friendship. The similarity between two communities is computed by finding the percentage of data points in common between them. The proposed framework converts a given dataset into snapshots, obtaining communities at each snapshot using the MCL algorithm. In the next step, several events, both in the community and entity level, are extracted using binary operations using similarity matrices. A similarity matrix is a structure filled with similarity values between communities. In the community level, the proposed framework extracts the following events: a) Continue, b) k-Merge, c) k-Split, d) Form and e) Dissolve. In the entity level, the framework extracts the following events: i) Appear, ii) Disappear, iii) Join and iv) Leave.

Additionally, the authors propose three indexes for the analysis of the performance on the entity level, namely:

- *Stability* indicates the degree of interactions with the same set of entities over a given time t.
- *Sociability* indicates the amount of interactions that an entity has with different entities over time.
- *Influence* indicates the amount influence that an entity has over others. For example, when entity x moves to another community, how many of the entities will follow x's lead?

In the community level, the authors propose an index for the measure of the popularity of a community at time t, named *Popularity index.*

The final contribution is a diffusion model for evolving networks. A diffusion model is a method from the field of complex networks that aims at identifying the key nodes of a social network for effective information propagation [5], [13]. The authors demonstrate that their approach can generate accurate predictions specially in the Sociability index.

When comparing the similarity function of this framework to the similarity function of MONIC and MEC, we observe that the former does not include the notion of spatial overlap nor data ageing. Moreover, the k value in the k-Merge and in the k-Split transitions are a threshold that should be respected in the context of the event to be considered valid, much like the match threshold in MONIC.

FacetNet [28] is a framework that uses a different approach to detect and analyze community evolution. Instead of analyzing the sequence of snapshots containing communities, it fits a temporal evolution model by computing two structures: a community network and an evolution network. To compute those structures they use the concept of *Q-Modularity* and unify the process of analyzing communities and evolution. They showed that their approach is more robust to noise. Moreover, the authors suggested that a dramatic chance in a short period of time is unlikely. Therefore, a temporal evolution that takes that information in account yields a descriptor that is more significant of a real evolution.

Although FacetNet considers the whole sequence of snapshots to find the description of the evolution, the analysis considers consecutive pairs of snapshots. Besides, the generation of the communities is done using soft clustering, meaning that a given data point can be part of multiple clusters, and weights each individual (equivalent of a data point) differently. FacetNet also provides a visualization tool that is similar the one proposed in MEC.

MODEC is a framework proposed to model transitions in the context of dynamic community mining [49], [50]. It takes a sequence of graphs and segments the input sequence into snapshots. The data within each snapshot is then transformed into a graph by executing a community detection algorithm. The authors of MODEC use the same terminology for transitions as [6], and their framework detects five types of transitions, namely: *form, dissolve, survive, split* and *merge.*

The overall MODEC framework is similar to MONIC and its derivatives. However, MODEC has a different similarity function and transition detection algorithm. Two communities are similar if and only if the intersection of the compared communities is $k\%$ proportional to the size of the largest of the two, where k is a user provided variable. The match detection however is done in an iterative form: if a community at time t does match with any community in $t-1$ then the algorithm iteratively searches past snapshots in a reverse order. If no match is found until t_0 then a form event is discovered w.r.t. the analyzed community.

All datasets used for MODEC evaluation were temporal textual datasets. Therefore, the authors use the topic continuity to evaluate the framework's performance, showing better results than most algorithms w.r.t. that metric. Finally, MODEC also uses a visualization tool that is similar to the one proposed by MEC authors.

Chapter 3

Methodology

The main goal of cluster monitoring is to *track data over time* with the intent of performing knowledge discovery w.r.t data behavior and data change. In this section I propose an approach to tackle the problem of change detection using a pattern-based cluster monitoring framework. I motivate this work in Section 3.1. I propose a pattern-based framework for pattern discovery in Section 3.2. Finally, I extended a transition-based framework for cluster monitoring in Section 3.3.

3.1 Motivation

Analyzing changes between two timestamps is the standard in the clustering monitoring field [6], [28], [39], [48], [50]. While transitions are a simple way to describe change in data, we hypothesize that changes might occur over multiple timestamps. Furthermore, it is hard to find values to the set of thresholds used in cluster monitoring frameworks so that we can find transitions that are meaningful [41].

One way to find changes that span multiple timestamps is to combine discovered transitions. This approach was initially developed by Ntoutsi et. al. [36] where they summarize survival transitions, but the authors argue that it is much harder to combine multiple split and merge transitions. Combining transition is even more challenging when thresholds are taken into consideration, since different thresholds might lead to different transitions. Lower threshold values might also increase the detection of noisy transitions, besides increasing the number of transitions detected.

Consider the following examples that illustrate the issues. The diagram in Figure 3.1 depicts a scenario where the circles represent clusters, the weights over the arrows represent similarity values and t_1 and t_2 represent two consecutive timestamps. We have two parameters that need to be set: τ, which represents a match threshold; and λ, which represents a split threshold. In these examples we follow the conventions of MONIC and MEC.

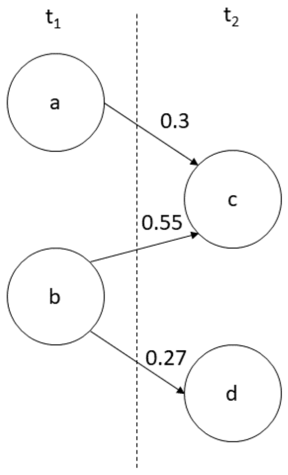


Figure 3.1: Diagram with clusters and its respective similarity values between two timestamps.

Example 1. When we set $\tau = 0.5$ and $\lambda = 0.25$, we find two transitions: a death transition for a, and b splits into c and d. The death transition is detected since the only cluster that a is similar to is c and the similarity is smaller than τ. The split transition is detected since the similarity between b and both c and d are larger than λ and the sum of these similarities is larger than τ.

Example 2. Using the same diagram and setting $\tau = 0.5$ and $\lambda = 0.3$ we find the following transitions: survival of b into c, death of a and birth of d. The survival is detected because the similarity between b and c is at least τ. The death is detected because the similarity of a to any cluster in t_2 is smaller than τ. Finally, the birth is detected since there is no cluster in t_1 which the similarity to d is larger than τ. No split is detected this time since $\lambda = 0.3$, preventing the similarity between b and d being taken in consideration.

Example 3. We set $\tau = 0.4$ and $\lambda = 0.3$ and detect two transitions: a and b merge into c; and the birth of d. The merge transition is found since both a and b have similarity values w.r.t. c that are at least τ. The birth transition is found since there is no cluster that is at least λ similar to d in t_1.

Example 4. We set $\tau = 0.4$ and $\lambda = 0.25$, detecting two transitions: a and b merge into c and d is born in t_2.

Comparing examples 1 and 2 we find that a change in λ results in transitions of different types and a change in the number of transitions detected. This example shows that the transition detection process is dependent on (and often quite sensitive to) the thresholds set. Example 3 and 4 detect the same transitions while having different parameters. In example 4, b could also split into c and d, however, because of the conventions established in the state-of-the-art in cluster monitoring, it is not possible to detect split and merge transitions simultaneously. One of the first works to address this problem was MODEC [50], where survival, splits and merges are not mutually exclusive.

To exemplify the issues of transition detection in multiple timestamps we use the diagram in Figure 3.2. We use the same conventions established by MONIC and MEC again where we have two thresholds τ and λ.

Example 5. We set $\tau = 0.5$ and $\lambda = 0.3$. In this case one split is detected where a splits into b and c. No split is detected from t_2 to t_3, since although the similarities from b to d and e sum to be greater than τ, the similarity between b and e is less than λ. The death transition of c is detected at t_2 and

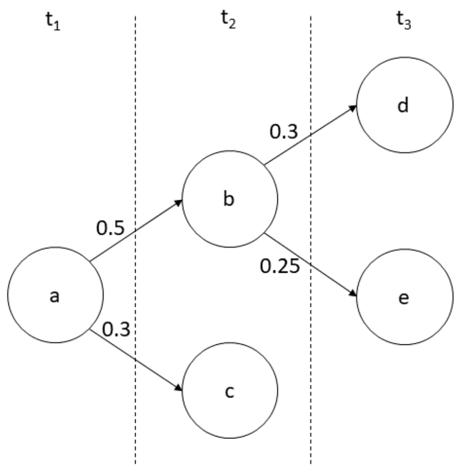


Figure 3.2: Diagram with clusters and its respective similarity values between three timestamps.

the birth transitions of d and e are detected at t_3.

Example 6. We set $\tau = 0.5$ and $\lambda = 0.25$. In this case two splits are detected: a splits into b and c; and, b splits into d and e.

When comparing examples 5 and 6 we observe that changing threshold values can cause a transition to not be detected. To tackle this problem, a change detection method that is robust to changes in similarity values is necessary.

Motivated by the problem of detection of change in datasets where the similarity between clusters is dynamic, we propose a change detection framework based on frequent patterns. With our framework we can detect frequent patterns of any size or shape that are frequent enough to be representative w.r.t. a minimum frequency.

For our next example, we set a labeling scheme to the graph edges where each node label is a tuple composed of the number of incoming edges and the number of outcoming edges. For instance, node a in Figure 3.2 is labeled $(0, 2)$, since it has no incoming edges and two outcoming edges. We also label the edges based on their values. We label edges with weight in the range $[0, \frac{1}{3})$ as *low*, $[\frac{1}{3}, \frac{2}{3})$ as *medium*, and $[\frac{2}{3}, 1]$ as *high*. For brevity we shorten these labels as l, m, and h, respectively. In Figure 3.3 we show the diagram from Figure 3.1 in (a) and the diagram from Figure 3.2 in (b). With our proposed framework we can distinguish between different kinds of *patterns* and present them as part of the knowledge discovery process.

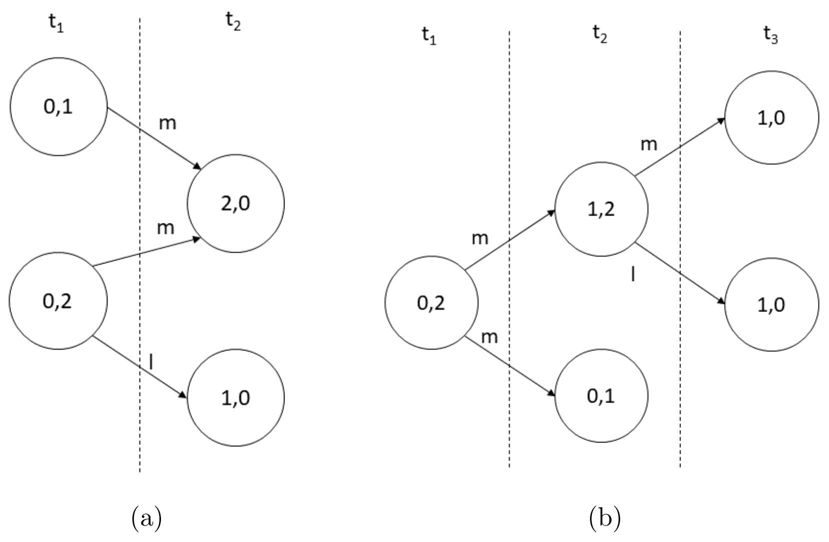


Figure 3.3: Labeled Diagram derived from figures 3.1 (a) and 3.2 (b)

Example 7. Let us search for all patterns that are composed of three nodes and span exactly two segments. In Figure 3.3 (a) we find the pattern where nodes $(0, 1)$ and $(0, 2)$ have each one edge with label m (medium) pointing at node $(2, 0)$. We also find the pattern where node $(0, 2)$ have two edges: one with label m pointing at node $(2, 0)$; and another with label l (low) pointing at node $(1, 0)$. These two patterns appear once in the diagram and are equivalent to a merge transition and a split transition, respectively. Moreover, using our framework both patterns can be detected simultaneously, which does not happen in the transition detection framework, as shown in **Example 4**.

Example 8. Let us search for patterns that are composed of at least four nodes and span at least three segments using the diagram in Figure 3.3 (b). We show the discovered patterns in Figure 3.4. Using a transitions-based approach, none of the patterns presented in Figure 3.4 can be discovered by a single transition. Moreover, when using a combination of transitions, these patterns cannot be discovered simultaneously without changing of thresholds τ and λ, as shown in **Examples 1-6**. In contract, our proposed pattern-based approach can discover all these patterns simultaneously, besides allowing us to find patterns based on frequency. We present our pattern-based framework in the next section.

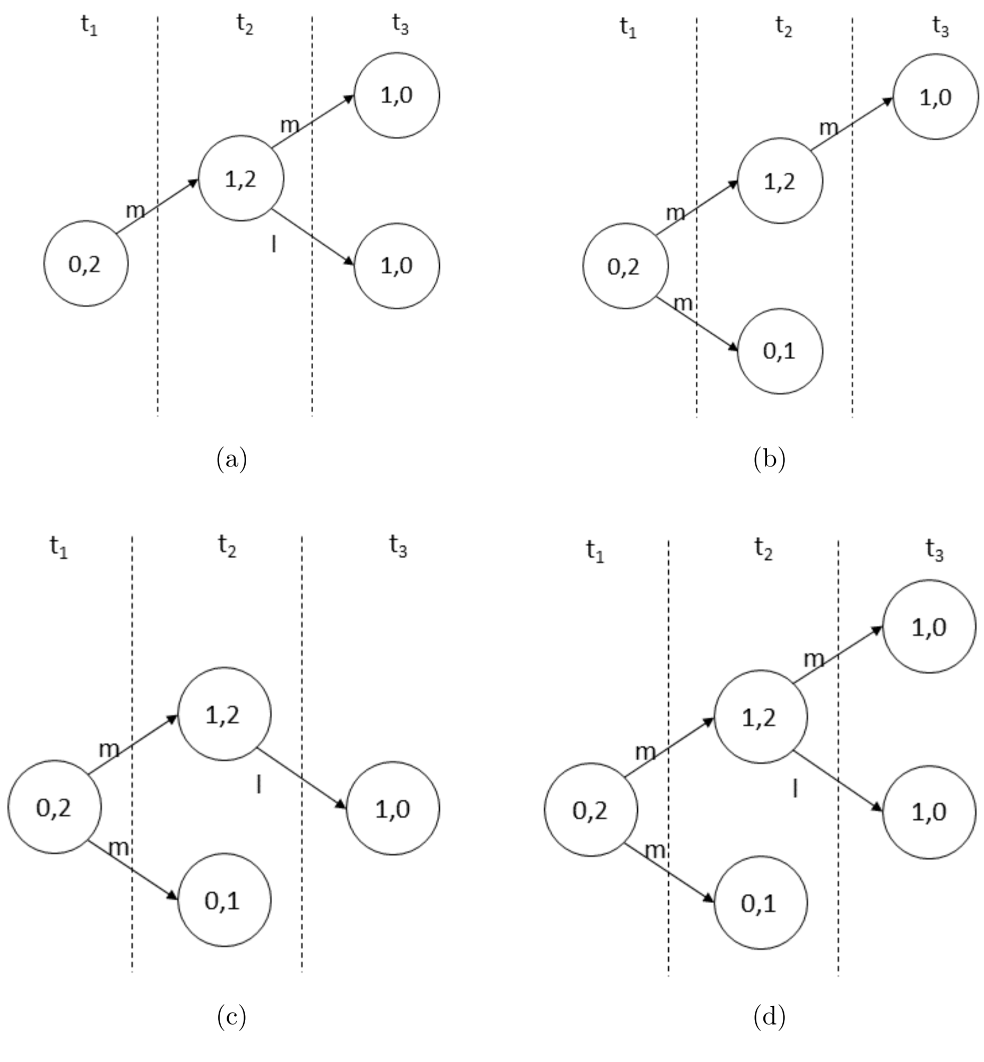


Figure 3.4: Patterns with four or more nodes and that span at least three segments found in the diagram in Figure 3.3 (b)

3.2 A pattern-based framework for cluster monitoring

In this section I propose a framework to track a set of data points over a time interval. To continuously match these sets, I use a notion of similarity between the data contained in each set. Tracking sets of points is an important task in in many fields, such as Social Sciences, Computer Vision, Social Network Analysis, among others. This task is challenging because the data collection might not be fast enough, the data might have high velocity profile, or it might be hard to find natural groups (clusters) in data that changes over time.

In this section I propose a framework for detecting patterns in temporal dataset that is pattern-based. Differently from the state-of-the-art transition-based methods, I propose to extract patterns that are frequent in the EG through mining *patterns*. This approach allows us to discover not just one-step patterns but patterns of any kind and shape that are frequent in the data. In datasets where there is a pattern that spans into multiple timestamps, it might be hard to detect with a transition-based approach.

The framework is composed of six steps:

1. collecting the data;
2. segmenting the collected data into smaller parts;
3. clustering each segment;
4. finding similarity values between the discovered clusters over time;
5. building an evolution graph from the clusters and similarities;
6. pattern mining using frequent subgraphs;

I discuss these steps in detail further in the next sections. In each section I first introduce a more general concept and then a specific temporal version of the given concept. This is because this thesis is focused on detecting patterns that have a temporal characteristic.

3.2.1 Segmentation

A dataset D is a collection of data points d_i, where I assume that the collected dataset is segmented into smaller parts, named *segments* (ζ). Data in a specific segment may, e.g., contain data from a specific time interval. We define the segmentation of a dataset D as follows:

Definition 3.2.1 (Segmentation)**.** A dataset segmentation, w.r.t. some segmentation criterion is a sequence of segments $\langle \zeta_0, \zeta_1, ..., \zeta_k \rangle$ where $\bigcup_{i=0}^{k} \zeta_i = D$.

I assume that a criterion of segmentation is given. For example, in a temporal dataset a segmentation criterion is a series of intervals, or in a geospatial dataset the segmentation criterion might be determined by geographic characteristics or along a spatial dimension.

Because I segment the dataset into a sequence $\langle \zeta_0, \zeta_1, ..., \zeta_k \rangle$ I assume that there exists a linear order $0, 1, ..., k$, such as time, in the dataset. The linear order of the segments is introduced so that: a) tracking of an evolution can be established w.r.t the linear order of the segments; b) clusters can be discovered within the segments instead of within the entire dataset.

3.2.1.1 Temporal Segmentation

Definition 3.2.2 (Temporal Dataset)**.** A Temporal Dataset is a dataset composed of tuples $p = (d, t)$, which we call elements, over a domain D with $d \in D$ and $t \in T$, where T is a set of timestamps. Given an element $p = (d, t)$ of a temporal dataset TD, we call t the timestamp of p or $timestamp(p) = t$, and we call d the data point of p, or $datapoint(p) = d$.

Differently from a regular dataset composed of only points, in a temporal dataset, for clarity, I make the distinction between the data and the timestamp associated to the data.

A segmentation of a temporal dataset can be obtained by partitioning the temporal dimension into intervals. This segmentation is beneficial because:

- analyzing smaller chunks of temporal information can give insights that are more meaningful for the covered time period;
- temporal datasets can be large, which requires batch processing, since they might not fit into memory all at once;
- the segmentation process enables the description of transitions.

The smaller parts that result from the segmentation are called *temporal segments* (Figure 3.5) and contain all elements of TD which timestamp falls within a given *time window*.

Definition 3.2.3 (Time window)**.** A time window is an interval $[t_{start}, t_{end}]$ defined by two timestamps t_{start} and t_{end} such that $t_{start} \leq t_{end}$. The window size Δ_w of a time window $w = [t_{start}, t_{end}]$ is given by:

$$\Delta_w = t_{end} - t_{start}$$

Definition 3.2.4 (Temporal Segment)**.** A Temporal Segment ζ^w of a temporal dataset TD with respect to a time window $w = [t_{start}, t_{end}]$ is defined as the set $\zeta^w = \{e \in TD : t_{start} \leq \text{timestamp}(e) \leq t_{end}\}$. The width of a segment ζ^w is the size of its time window, Δ_w.

Definitions 3.2.3 and 3.2.4 allow us to describe a temporal dataset as a sequence of consecutive temporal segments $\langle \zeta_0^{w_0}, \zeta_1^{w_2}, \zeta_2^{w_2} ..., \zeta_k^{w_k} \rangle$.

Definition 3.2.5 (Temporal Segmentation)**.** A temporal dataset segmentation, w.r.t. a time window size Δ, is a sequence of temporal segments $\langle \zeta_0^{w_0}, \zeta_1^{w_1}, \zeta_2^{w_2} ..., \zeta_k^{w_k} \rangle$, where $k > 1$ and $w_i = [t^i_{start}, t^i_{end}]$, satisfying the following the conditions:

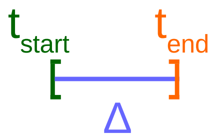


Figure 3.5: An illustrated temporal segment and its elements. In green the t_{start}, in blue the time window size Δ and in orange the t_{end}.

- $\bigcup_{i=0}^{N} \zeta_i = TD$;
- the width of every temporal segment ζ_i is Δ, i.e. $\Delta_{w_i} = \Delta$;
- $t_{start}^{i+1} = t_{end}^{i} + 1$, for all $0 \leq i < k$.

The sequence that results from a segmentation consists of non-overlapping elements (Figure 3.6), since t_{start} for all elements must be different, and $k > 1$. This is a special case of non-overlapping intervals where all intervals have the same size and the union of all intervals cover the entire timeline.

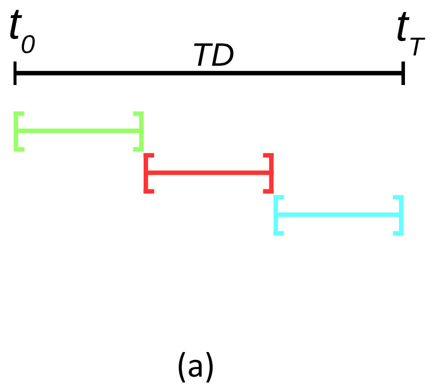


Figure 3.6: Example of a segmentation of a temporal dataset TD.

The proposed approach consists of making the length of all temporal segments w.r.t. its time windows equal. This condition guarantees that we analyze consistent periods of time. It also makes the analysis and visualization simpler, since we are looking at equal time periods. On the other hand, this approach may not be efficient in data streams where the data does not arrive continuously, since we might find segments with too much, too little or no data at all. Such scenarios might require a data-oriented approach, where the dataset is segmented according to the amount of data collected. While the data-oriented approach provides a solution for such streams, it might not provide consistent time periods which makes the analysis more difficult. In this work we focus on a time-consistent segmentation approach.

3.2.2 Clustering

So far, I have separated the data into smaller segments. Further, I compute the clusters within each segment with the goal of simplifying the tracking of the data over time. I define a clustering as follows:

Definition 3.2.6 (Clustering)**.** A clustering ξ of a segment ζ is a partition of ζ into k non-empty clusters or k non-empty clusters plus a set of outliers, i.e., $\xi = \{C_1, C_2, ..., C_k\}$ or $\xi = \{C_1, C_2, ..., C_k\} \cup \{C_0\}$ (where $\{C_0\}$ is the set of outliers), so that:

- $\forall C_i \forall C'_j [(C_i \in \xi \wedge C'_j \in \xi \wedge C_i \neq C'_j) \rightarrow C_{t_i} \cap C'_j = \varnothing]$;
- $\bigcup_{k=0}^{|\xi|} C_k = \zeta$;
- some clustering criterion is satisfied, e.g. the members of each cluster are more similar to each other than to the other data records, or they form a density connected set.

I will be using hard clustering in this framework, i.e., each data point is part of only one cluster or is an outlier. Moreover, the clusters and the outliers partition the whole set of points in partition ζ. Lastly, the clustering needs to satisfy some clustering criterion, meaning that the partition is not “random”. For example, in k-means the optimization is to find the optimal locations of the centroids.

One distinct characteristic of our definition is that we allow a group of outliers in the clustering space. Groups of outliers are only generated in specific types of clustering, e.g. density-based clustering. In the state-of-the-art methods for tracking sets of data we find that the group of outliers is removed. However, by removing this group of data points we might be discarding valuable information, such as shifts in the data points over the space and the influence of outliers over the remaining data points.

The applied clustering algorithm has influence in the process of pattern discovery because of two main factors:

1. distinct clustering techniques have different assumptions about the data and generate different cluster representations;
2. distinct clustering techniques might return different clusterings.

Factor 1 influences the computation of similarity values, since cluster representations might be used by a similarity function. Factor 2 inherently changes the pattern discovery process, since a different clustering might lead to different patterns to be detected.

3.2.3 Similarity

The clustering process results in a sequence of segments with data which is now divided into groups. The data within each group is similar among themselves given a clustering criterion. The framework computes the similarity between all pairs of clusters (C, C'), where $C \in \xi_i$ and $C' \in \xi_{i+1}$. A *similarity function* is used to compute the similarity value:

Definition 3.2.7 (Similarity Function)**.** A similarity function is a function that computes a measure of relatedness, e.g. similarity, between two sets of data points C and C' where all points in C and all points in C' are part of a dataset D, i.e. $\textsc{Sim}(C, C') = w : w \in \mathbb{R}_{\geq 0}$. Moreover, the similarity function is symmetric, i.e. $\textsc{Sim}(C, C') = \textsc{Sim}(C', C)$.

In other words, the framework uses a symmetric function that returns non-negative values. I chose to use only symmetric functions to be able to use the same similarity in reverse order w.r.t the sequence of segments, allowing for tracking sets of data both forward and backwards.

An example of a similarity function is the Jaccard Coefficient:

$$Jaccard(C, C') = \frac{C \cap C'}{C \cup C'},$$

which is commonly used to measure similarity between two sets of data. The framework uses a similarity function to compute the λ-*similarity* between two sets of data points.

Definition 3.2.8 (λ-Similarity)**.** Two sets of data points C and C' are λ-similar if and only if $Sim(C, C') > \lambda$, where $\lambda \geq 0$ is a relatedness threshold.

I use the concept of λ-similarity to express the relatedness between two sets. Moreover, to discover whether two sets are indirectly related I use a λ-similarity chain:

Definition 3.2.9 (λ-Similarity chain)**.** A λ-similarity chain between two clusters C_{start} and C_{end} (denoted as λ-$Sim(C_{start}, C_{end})$) is a sequence of clusters $W = \langle C_1, C_2, ..., C_{j-i} \rangle$ such that $C_i \in \xi_i$ and C_i is λ-similar to C_{i+1} for $1 \leq i \leq j - i$.

With these definitions, we establish the following relationships between two distinct sets C and C' given a value for λ.

- **Related:** λ-$related(C, C')$ iff $|\lambda\text{-}\textsc{Sim}(C, C')| \geq 1$;
- **Directly Related:** $related_{direct}(C, C')$ iff $|\lambda\text{-}\textsc{Sim}(C, C')| = 1$;

- **Transitively Related:** $related_{transitive}(C, C')$ iff $|\lambda\text{-}\textsc{Sim}(C, C')| > 1$;
- **Ancestor and Descendant:** a) $\lambda{-}ancestor(C, C')$ iff $\lambda\text{-}\textsc{related}(C, C') \neq \emptyset$. b) $\lambda - descendant(C', C)$ iff $\lambda - ancestor(C, C')$.

Henceforth, I will refer to λ-*related*(C, C') as *related*(C, C') when $\lambda = 0$. Combining the order of the segments and the relatedness I establish a relatedness hierarchy between sets in different segments.

3.2.4 Evolution Graph

Definition 3.2.10 (Evolution Graph)**.** Given a segmented dataset D, where each segment ζ_i has been clustered resulting in a clustering ξ_i, we have a sequence of clusterings $\xi_i, \xi_{i+1}, ..., \xi_n$. An evolution graph $EG_{\xi_i,...,\xi_n}$ is a directed acyclic weighed k-partite graph $EG(V, E, w)$, where w is a function $w : E \to \mathbb{R}_{\geq 0}$ assigning a weight to each edge. An evolution graph satisfies the following conditions:

- there is a node $v \in V$ for each cluster $C \in \xi_i$, where $1 \leq i \leq n$
- there is a node $v \in V$ for each outlier group in ξ_i, where $1 \leq i \leq n$, if any
- there is a directed edge $e_{v,v'}$ for all related clusters in consecutive clusterings ξ_i and ξ_{i+1}, i.e.,

$$\begin{aligned} &e_{v,v'} \in E \text{ iff} \\ &1)\ v \in \xi_i, and \\ &2)\ v' \in \xi_{i+1}, and \\ &3)\ related(cluster(v), cluster(v')), \end{aligned}$$

  where $cluster(x)$ represents the cluster C correspondent to node x.
- for all $e_{v,v'} \in E$, $w(e_{v,v'}) = Sim(cluster(v), cluster(v'))$

The EG contains information about the data evolution in the dataset w.r.t. the proposed linear order of the segments and the clustering. It expresses similarities between clusters in neighboring segments and that can be tracked along the linear order.

3.2.5 Tracking Sets

Tracking a set of data points is an important task in many fields, however, most of the past literature in cluster monitoring focuses on identifying change in the data as a whole, not focusing on tracking the changes happening to a single set. In this thesis, besides identifying general trends, I propose a filtering to the evolution graph for tracking a single set of data points. Let Q be a set of data points that one is interest in tracking in the EG, which I call a *tracked set*:

Definition 3.2.11 (Tracked set)**.** A tracked set is a set of data points $Q = \{d_1, d_2, ...d_n\}$ where $Q \subseteq \zeta$ and where ζ is a single segment in D.

The framework restricts the origin of a tracked set to one segment in order to follow the linearity criterion present in a sequence. A tracked set can be a subset of one or more clusters, as long as these clusters are within the same segment. The problem of cluster tracking is a special case of set tracking, where the tracked set is equal to one of the clusters in a given segment.

A filtered EG w.r.t. Q can be derived from the EG that was built from the whole dataset:

Definition 3.2.12 (Q-Evolution Graph)**.** A filtered evolution graph w.r.t. Q, $EG_Q(V', E', w')$, is an evolution graph that contains only nodes that are related to Q. An $EG_Q(V', E', w')$ is computed from an evolution graph $EG(V, E, w)$

as follows:

$$V' = \{v \in V : ancestor(Q, cluster(v)) \vee descendant(Q, cluster(v))\} \cup \{v_Q\}$$

$$\begin{aligned} E' = &\{e_{v,v'} \in E : v, v' \in V'\} \cup \\ &\{e_{v,v_Q} : cluster(v) \in \xi_{i-1}\} \cup \\ &\{e_{v_Q,v} : cluster(v) \in \xi_{i+1}\} \end{aligned}$$

$$w'_{e_{v,v'}} = Sim(cluster(v), cluster(v'))$$

Where v_Q is a node representing the set Q.

In other words, the set of nodes V' is composed of nodes that correspond to clusters that are either ancestors or descendants of a set Q plus a node that represents Q. The set of edges E' is composed of all edges that connect to nodes that are either ancestors or descendants of Q and the new edges between the direct ancestors to Q and between Q and the direct descendants of Q. An example of an EG_Q is shown in Figure 3.7.

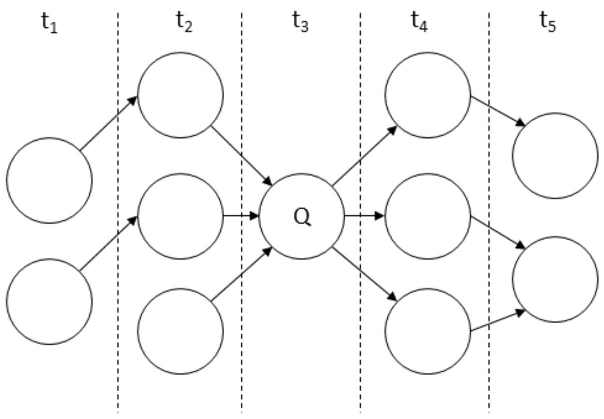


Figure 3.7: Example of an EG, nodes that precede the tracked set Q point to Q, and Q points to the sets that succeed it.

With a general EG it is possible to discover information about the whole dataset through careful inspection of the graph. However, more detailed insights can be extracted from a filtered evolution graph EG_Q. For instance, it is possible to separate the EG_Q into a *backward-tracking graph*, which consists of nodes that are ancestors of a tracked set Q and Q itself; and a *forward-tracking graph*, which consists of nodes that are descendants of a tracked set Q and Q itself.

3.2.6 Pattern Discovery

The problem of pattern discovery in graph structured data consists finding a pattern that is frequent in either a graph composed of a single connected component or a graph that is composed of multiple, smaller connected components. The later problem has larger attention from the graph mining community since it can be solved as a frequent transaction problem [27]. The problem of pattern discovery in graphs is known to be NP-Complete, while some instances are GI-Complete[1], when it is necessary to prove that two graphs are isomorphic [12]. To the best of our knowledge, most of the literature in graph pattern discover is based on labeled graphs [26], [55]. The only known exception is [25], where the authors use relabeling to discover topological patterns in graphs.

Given an evolution graph following the same concepts introduced in Definition 3.2.10, our goal is to mine patterns that are frequent in an evolution graph or a collection of Q-evolution graphs. We define a frequent pattern as follows:

Definition 3.2.13 (Frequent Pattern)**.** Given a set of evolution graphs G, which could be a singleton containing just a single an evolution graph $EG(V, E, w)$ or a collection of Q-evolution graphs, a pattern p is a graph $G'(V', E', w')$ and that consists of a single connected component and that occurs in the elements of G more frequently than a threshold ϵ.

In other words, a frequent pattern is a subgraph of the evolution graph that occur frequently given a threshold.

[1]Graph-Isomorphism Complete.

Since the EG is computed from clusters, we can consider that: a) the EG is not labeled; or, b) the EG is uniquely labeled, i.e. each node $v \in V$ has a unique label. Since the EG is a DAG, we propose a labeling function based on the in and out degree of a node. The in-degree of a node v is the number of edges that are incident *to* v, i.e. the number of edges that point to v and is denoted $d^{-}(v)$. Similarly, the out-degree of a node v is the number of edges that are incident *from* v, i.e. the number of edges that point away from v and is denoted $d^{+}(v)$.

A labeling function is based on a relation $m \subseteq V \times V'$ between the nodes of two graphs $G(V, E)$ and $G'(V', E')$.

We define the node labeling function as $f : V \rightarrow \mathbb{N} \times \mathbb{N}$ where

$$f(v_i) = (d^{-}(v_i), d^{+}(v_i)), \tag{3.1}$$

where $d^{-}(v) \in \mathbb{N}$ is the in-degree of a node and $d^{+}(v) \in \mathbb{N}$ is the out-degree of a node. The result is a tuple composed of the in-degree and out-degree of the node. Because in-degree and out-degree are invariants of a DAG, the labeling function defined above can be used to label (or re-label) a graph without changing its structure.

We define the edge labeling function as $g : E \rightarrow \mathbb{N}$, where

$$g(e_{v_i,v_j}) = l^I \text{ iff } l^I_{min} < w(e_{v_i,v_j}) \leq l^I_{max}.$$

where l^I is a label, and $I = [l^I_{min}, l^I_{max}]$ is an interval in a set of intervals that partition the range of values of w. $l^I_{min} \in \mathbb{R}$ is the minimum value of a range l^I and $l^I_{max} \in \mathbb{R}$ is the maximum value of a range of weights that will imply the same label. With this definition we can create several ranges of values and label edges w.r.t. these ranges.

Example. Let $G(V, E, w)$ be an evolution graph where $V = \{a, b, c, d\}$, $E = \{e_{a,b}, e_{a,c}, e_{b,d}\}$ and $w(e_{a,b}) = 0.2, w(e_{a,c}) = 0.4, w(e_{b,d}) = 0.7$. We illustrate the described graph in Figure 3.8 (a). We use our definitions above to label this graph, where

$$g(e_{v,v'}) = \begin{cases} 0, \text{if } 0.0 < w(e_{v,v'}) \leq 0.3 \\ 1, \text{if } 0.3 < w(e_{v,v'}) \leq 0.6 \\ 2, \text{if } 0.6 < w(e_{v,v'}) \end{cases} \tag{3.2}$$

The evolution graph $G(V, E, w)$ is then mapped into a graph $G'(V', E', w')$ where $V' = \{(0,2), (1,1), (1,0), (1,0)\}$, $E' = \{e_{(0,2),(1,1)}, e_{(0,2),(1,0)}, e_{(1,1),(1,0)}\}$ and $w'(e_{(0,2),(1,1)}) = 0, w'(e_{(0,2),(1,0)}) = 1, w'(e_{(1,1),(1,0)}) = 2$. The labeled graph is illustrated in Figure 3.8 (b).

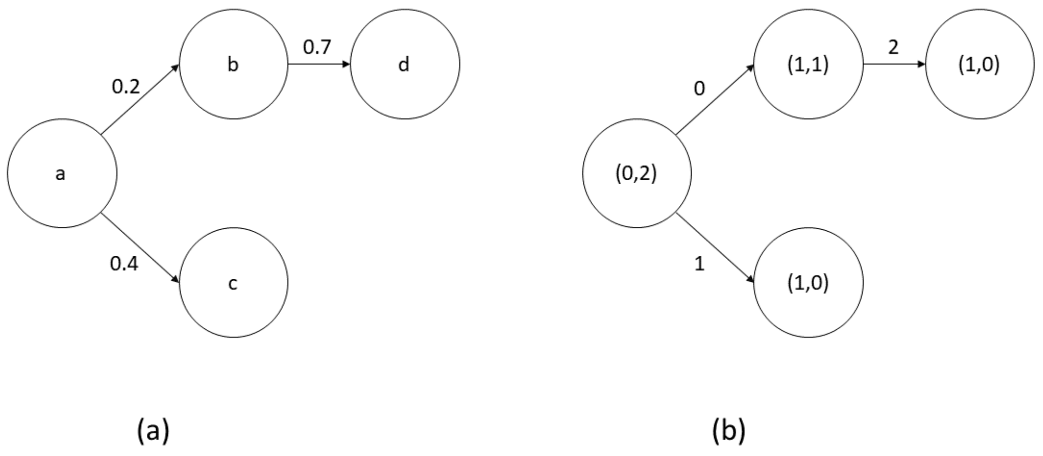


Figure 3.8: Two isomorphic graphs, where (a) is the initial EG that is labeled into (b).

With the new labeling we can mine the graph for patterns using a subgraph mining algorithm that finds frequent patterns, if they exist, in the evolution graph. Most subgraph detection algorithms assume that the input is a set of connected components, in which they simply count the occurrence of the mined patterns [24]. This is because these algorithms detect subgraphs based on a support measure. Given a set of connected components C_G of a graph G, support is defined as:

$$supp(x) = \frac{|\{c \in C_G : x \subseteq c\}|}{|C_G|},$$

in words, the support of a subgraph x is the count of connected components c that contains x divided by the number of connected components in G.

A naive way to search for subgraphs is to generate all candidate subgraphs and search for them in the evolution graph. One can use an Apriori-based approach [23] to detect frequent patterns, where patterns are modeled as rules and connected components are modeled as transactions. Most algorithms for subgraph detection, however, rely on a heuristic search, since the search space is huge [24].

Evaluating knowledge extracted by an automatic process remains an open challenge in the field of knowledge discovery. Many metrics have been studied in this context and there is a general agreement that most of the evaluation depends on the user's goal [21].

In [32] the authors define a taxonomy of interestingness measures, where they discuss that a pattern can be deemed interesting based on its complexity and how representative of the data it is. In contrast, they show that a pattern can also be "unexpected" and this interesting, meaning, however, that it is not frequent but is rather an outlier.

Other interestingness measures, such as "actionable" or "novel", are considered subjective, since they require background knowledge from the user. Moreover, such measures constrain the discovery process to what the users anticipate or hypothesize.

In this thesis we do not claim that the patterns created by our framework are "interesting" in terms of subjective measures. We do however evaluate the data based on support, meaning that our patterns are frequent w.r.t. some user defined minimum support.

3.2.6.1 Tracking Sets and Pattern Discovery

Using definition 3.2.12 we can create an EG_Q w.r.t. an external criterion, i.e. a criterion that was not used for computing the clusters or the similarity values of the EG_Q. With multiple criteria, we can create a collection of Q-evolution graphs. We can then use this collection of Q-evolution graphs to search for patterns that are common between different filtered graphs. For example, we can select different nodes in the same segment which may result in different Q-evolution graphs.

Instead of using different criteria in the same segment, we can also derive multiple Q-evolution graphs using the same criterion in different segments. With the generated Q-Evolution Graphs we can compute the trends of the same criterion when applied to different segments of the data. For example, we can select a node in each segment and filter the evolution graph with respect to the nodes that are similar to the select node. The filtered Q-evolution graphs can then be mined for frequent patterns using a subgraph mining algorithm.

3.2.7 Discussion

The framework proposed in this section is substantially different from the literature in cluster monitoring. This is because, to the best of our knowledge, we are the first to successfully propose a cluster monitoring framework to detect multi-step temporal patterns of arbitrary length.

While transitions are a simple way to describe changes in data, there might exist more complex patterns that might explain the behavior of data in a more comprehensive way. Our proposed approach detects those patterns through the modelling of the temporal data into a graph that is then labeled according to a node's degree. This labeling then allows us to mine a graph for patterns using a subgraph mining algorithm.

3.2.8 Summary

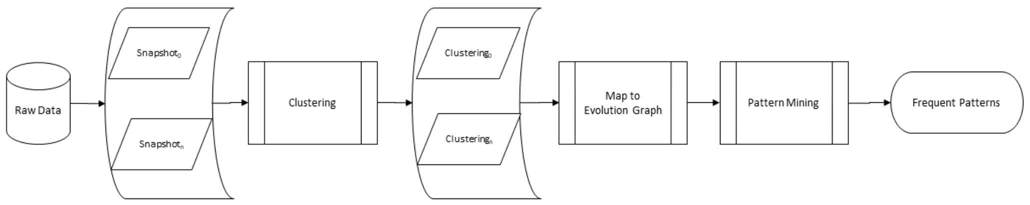


Figure 3.9: Diagram showing the pipeline of the pattern-based framework.

In summary the pattern-based framework has six steps: data collection, segmentation, clustering, extraction of similarity values, computing evolution graph and pattern discovery. A diagram containing the steps and their order is shown in Figure 3.9.

3.3 A transition-based framework for cluster monitoring

In this section I describe a transition-based framework. It is composed of a pipeline that allows for the generalization of existing cluster monitoring approaches using transitions as its main tool to detect change.

In this section I assume that a dataset was collected, segmented, clustered and an evolution graph was computed. With this evolution graph we can perform the process of transition extraction, which I discuss further.

3.3.1 Similarity and transition extraction

Cluster transitions are indicators of change in this approach, showing whether clusters involved in the transitions are changing or not. The proposed method extracts the following external transitions: *birth, death, split, merge* and, *survival*. Moreover, it extracts the following internal transitions: *expansion, shrinkage, diffusion, contraction* and, *shift*. External transitions are those that affect the clustering space. Internal transitions are those that affect the cluster internal aspects. I define two thresholds to be used in the transition detection: i) τ is the threshold used to detect higher similarity between two clusters; and, ii) λ to detect relatedness between two clusters.

The external transitions are defined as follows:

- **Survival:** a cluster $C \in \xi_{t_i}$ has a similarity value greater than or equal to τ with only one cluster $C' \in \xi_{t_{i+1}}$:

$$survival(C, \xi_{t_{i+1}}) \text{ iff } |\{C' \in \xi_{t_{i+1}} : Sim(C, C') > \tau\}| = 1$$

- **Birth:** a cluster has no ancestors:

$$birth(C, \xi_{t_{i-1}}) \text{ iff } \{C' \in \xi_{t_{i-1}} : Sim(C, C') > \tau\} = \emptyset$$

- **Death:** a cluster has no descendants:

$$death(C, \xi_{t_{i+1}}) \text{ iff } \{C' \in \xi_{t_{i+1}} : Sim(C, C') > \tau\} = \emptyset$$

- **Split:** a cluster $C \in \xi_{t_i}$ is related to multiple clusters $C'_k \in \xi_{t_{i+1}}$ by at least λ and the sum of the similarity values is greater than τ, where λ is called the split threshold and τ is called the match threshold:

$$\begin{aligned} & split(C, \xi_{t_{i+1}}) \text{ iff} \\ \exists Z : {} & 1)\ Z \subseteq \xi_{t_{i+1}}, and \\ & 2)\ \forall C' \in Z : Sim(C, C') > \lambda, and \\ & 3)\ \sum_{C' \in Z} Sim(C, C') > \tau \end{aligned}$$

- **Merge:** a set of clusters $Z = \{C_k \in \xi_{t_i}\}$ have similarity greater than τ

to a single cluster $C \in \xi_{t_{i+1}}$:

$$\begin{aligned} &merge(Z, \xi_{t_{i+1}}) \text{ iff} \\ &Z \subseteq \xi_{t_i}, and \\ &\exists C' : 1)\ C' \in \xi_{t_{i+1}}, and \\ &\quad 2)\ \forall C \in Z : Sim(C, C') > \tau \end{aligned}$$

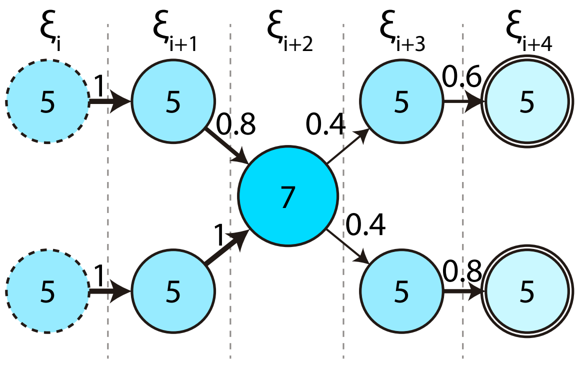


Figure 3.10: A toy evolution graph illustrating transitions

Example. I illustrate external transitions in Figure 3.10. In ξ_i, the nodes with dashed outlines indicate a *birth* transition. In ξ_{i+1}, the nodes with solid outline are part of a *merge* transition into the only node in ξ_{i+2}. In ξ_{i+2}, the single node *splits* into two nodes in ξ_{i+3}. The two nodes in ξ_{i+3} *survive* into the nodes in ξ_{i+4}. Finally, a *death* transition happens for each node in ξ_{i+4}, indicated by a double solid line.

Internal transitions are those that affect the internal structure of the clusters. Let $C \in \xi_{t_i}$ be a cluster that survives as $C' \in \xi_{t_{i+1}}$, we define the following internal transitions:

- **Count-increase**: The number of data points in a cluster $C' \in \xi_{t_{i+1}}$, is larger than the number of data points in $C \in \xi_{t_i}$:

$$count_{+}(C, C') \text{ iff } \ |C| < |C'|$$

- **Count-decrease**: The number of data points in a cluster $C' \in \xi_{t_{i+1}}$, is smaller than the number of data points in $C \in \xi_{t_i}$:

$$count_{-}(C, C') \text{ iff } |C| > |C'|$$

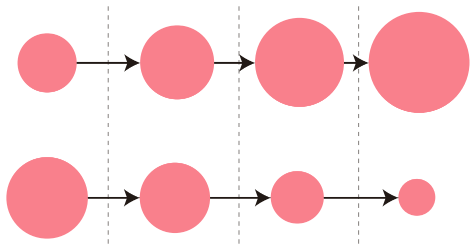


Figure 3.11: Example of count change transition. Size indicates the cardinality or a cluster, the larger the node the higher the cardinality. On top a $count_{+}$ transition, in the bottom a $density_{-}$ transition.

- **Density-increase:** the data distribution in C is less dense than in C', where $Den(X)$ is some density measure:

$$density_{+}(C, C') \text{ iff } Den(C) < Den(C')$$

- **Density-decrease:** the data distribution in C is denser than in C', where $Den(X)$ is some density measure:

$$density_{-}(C, C') \text{ iff } Den(C) > Den(C')$$

- **Shift**: the representation of a cluster C changed position in the clustering space w.r.t C', where $Repr(X)$ is a cluster representation and $Dist(X)$ is a function that measures the distance between two cluster representations:

$$shift(C, C') \text{ iff } Dist(Repr(C), Repr(C')) > 0$$

The count and density transitions are directly related, as the density is measured by the ratio of data points per volume of a cluster representation.

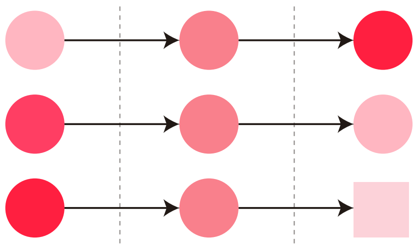


Figure 3.12: Example of density change transition. Color indicates the density, the darker the denser. On top a $density_{+}$ transition, in the middle a $density_{-}$ transition and, in the bottom an extreme $density_{-}$ going from dense to outlier group (represented by a square).

3.3.2 Discussion

The proposed transition framework has similar foundations to the ones found in the previous literature [41], [48]. We however propose a more robust flexible framework since the current state-of-the-art: a) uses non-symmetric similarity functions; b) allow for more than one cluster to match; and c) does not allow for a cluster to participate in possible non-mutually exclusive transition, e.g. split and merge simultaneously.

This flexible definition generalizes the current proposed frameworks in cluster monitoring. Furthermore, we extend them by using outliers in the process of knowledge discovery.

3.4 Final remarks

In this chapter I presented an approach to the problem of monitoring clusters using frequent patterns as indicators of change. In the context of temporal cluster monitoring, I proposed a pattern-based framework with focus on mining frequent patterns of arbitrary size, in contrast to the state-of-the-art that mines transitions. I also extended the cluster monitoring frameworks proposed in MONIC and MEC to use any type of symmetric function and to detect concurrent transitions.

Chapter 4

Experiments

We have experimented with two real-world datasets: The NSERC research proposals dataset[1] and the S&P500 dataset. We started our experiments with the NSERC dataset, where we first analyzed the data and then experimented with MONIC and our pattern-based framework. We then experimented with the S&P500 dataset, a larger dataset with stock prices over the years. Further we applied MONIC and our framework and showed that our pattern-based framework detected both transitions-like and complex patterns that would not be discovered with a traditional approach.

Our implementation was done using Python with clustering algorithms from Scikit-Learn Library [42] and the implementation of HDBSCAN by McInnes et al. [33]. We implemented our visualizations using GraphViz [16].

4.1 NSERC Dataset experiments

In this set of experiments, we used a dataset of research proposals that were funded by Natural Sciences and Engineering Research Council (NSERC), Canada's largest scientific research funding agency. Firstly, we tried to identify topics that are recurrent and that, through transitions, evolve into other topics over time. Second, we investigated if the funding granted by NSERC was diversi-

[1]Part of the results of this experiment have been reported in a late CASCON 2019 paper

fied. Lastly, we use MONIC and our proposed framework to detect patterns in the data and performing a comparison between the two approaches.

All the experiments described in this section were implemented using the Python3 language, Scikit-Learn [42] for general machine learning tasks, NLTK [29] for Natural Language Processing (NLP) related tasks, and GraphViz [16] for drawing visualizations. For HDBSCAN*, we used the implementation by McInnes et al. [33], [43].

4.1.1 Dataset

The dataset was publicly available on NSERC's website[2] and was collected by researchers from the software engineering laboratory at the University of Alberta. The dataset contained funded proposals submitted between 1991 and 2018, from several areas of knowledge adjudicated by 270 committees from 4014 institutions.

For the purpose of this research, I have narrowed down the dataset to only Computing Science themed proposals. And, because many proposals prior to 2006 did not contain metadata, including the research proposal, I used only proposals from 2006 to 2017. The proposals were written in English or French. Initially I have experimented using automatically translated proposals, but that approach showed to be too noisy since the directly translated words were not always related to the text meaning. Therefore, I discarded all proposals where the original text is written in French. The final dataset contained 1253 proposals written in English and adjudicated by eleven different committees over eleven years.

[2]https://tinyurl.com/y3bcojmu

4.1.2 Experiment Overview

In this experiment I only used the research summaries to represent each proposal. These summaries are short texts up to one page that briefly describe the proposed work. Since this was textual data, I had to represent the data in a form that it can be clustered. Common approaches are bag-of-words, term-frequency and tf-idf vectors, that are then clustered using algorithms such as K-means [3], [4]. I chose tf-idf as the representation of the documents, since the other approaches are simple counts of a collection of the words contained in each document. In contrast, tf-idf takes into account the frequency of each word in the document and the frequency of that same word across all documents, then weighing words according to the amount of information it provides. The tf-idf value is calculated as follows:

$$\mathit{tfidf}(t,d,D) = \mathit{tf}(t,d) \times \mathit{idf}(t,D) \tag{4.1}$$

where:

$$\mathit{tf}(t,d) = f_{t,d} \tag{4.2}$$

and:

$$\mathit{idf}(t,D) = -\log \frac{|d \in D : t \in d|}{N} \tag{4.3}$$

Where f is the term frequency, or raw count, in a document. The result is a vector with n dimensions where n is the size of the vocabulary of the document. Because the algorithm needs to calculate similarities, I defined n as the length of the vocabulary of the whole dataset.

I then segmented the data w.r.t. the year that the proposal was submitted, creating a total of eleven segments. I repeated the process of weighting the words by tf-idf values using only the documents within a segment. This approach provided more accurate weighting w.r.t. the proposals in the year that they were submitted. I then used HDBSCAN* to cluster the documents within each segment. I computed the mean of the tf-idf vector of each cluster to be its representation. I compared clusters in two subsequent segments pairwise, using the Cosine Similarity measure. I then computed an EG where each cluster corresponded to a node and there was an edge with weight w for each pair of nodes in consecutive years if their similarity is non-zero. This evolution graph was then analyzed by hand for insights. To aid in the visualization, I developed a tool that plots each cluster as a word cloud in the visualization of a node in the EG.

Additionally, because I wanted to discover frequent patterns, I labeled each node w.r.t. its in-degree and out-degree. Finally, I executed the *gSpan* algorithm [55] over the labeled evolution graph to search for frequent patterns that could be interesting to explore.

4.1.3 Data Processing

Transforming the data into a representation required that we cleaned the data to create a representation that was less likely to contain noise. In the cleaning process, punctuation and stop words were filtered out from the dataset. Stop words are terms that are frequent but most likely not meaningful, such as the word "the". The NLTK collection of English stop words was used in this process. We also identified that the words "research" and "use" were frequent and high-ranked in these documents, receiving high tf-idf scores. However, since they were not specific to proposals, we included both mentioned words in the list of stop words, removing them from the data.

After the data was free from stop words, each word was tagged using a Part-of-Speech Tagger and then lemmatized. Lemmatization is the process of transforming each word to its lemma, or dictionary form, by removing any inflections, e.g. “eating” and “ate” become the lemma “eat”. A Part-of-Speech Tagger (POS Tagger) is an analysis tool that labels each word automatically with a token such as “noun”, “verb”, “adjective” or “adverb”. Because the words were tagged, the lemattizer had more information, performing better than on untagged text [34]. In this process I used the NLTK POS and WordNet lemmatizer, part of the NLTK toolkit. The final dataset vocabulary was composed of 12286 words.

4.1.4 Document representation

I have explored document representations such as bag of words and tf-idf. The simplest form of a bag of words is the set of words that appear in a given document. Another approach is a bag of words with frequency counts, which is represented by a vector where each word represents a dimension and its value is the raw count of occurrences of that word in the document.

Another representation of a document is to transform this document into a term-frequency inverse-document-frequency (tf-idf) vector. This kind of representation weighs the words based on the amount of information provided. It first computes the raw frequencies of each word given a document. Then, it computes the negative logarithm of the number of documents divided by the number of documents where the computed word occurs. The final value is the product of the term-frequency and the inverse document frequency values. This representation was chosen to find words that were more meaningful w.r.t. a document, with an elegant and simple computation [44]. Finally, a group of documents was represented by the mean of the tf-idf vector of each document in the group.

We computed the tf-idf vectors of all documents in the data, and then computed the mean vector of these documents, creating an average summary of the tf-idf vectors w.r.t the whole dataset. Finally, we plotted the mean tf-idf vector as a word cloud (Figure 4.1). We observed that general Computing Science terms, such as "data", "model" and "system", had high tf-idf values. However, terms that are specific to sub-fields of Computing Science, such as "wireless" and "network" had surprisingly high tf-idf scores as well.

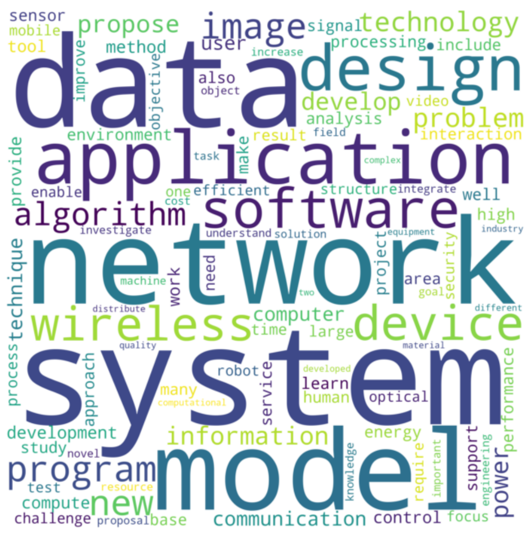


Figure 4.1: A word cloud w.r.t. the tf-idf representation of the whole data.

4.1.5 Segmentation

The dataset was segmented w.r.t. the competition year, creating eleven segments that contained the proposals that were funded in a given year. On average, each segment contained 104 documents, with a minimum of 48 documents in 2006 and a maximum of 237 documents in 2017.

4.1.6 Clustering

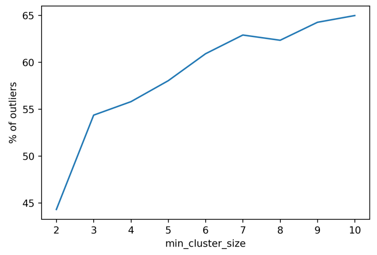
The documents within each segment were clustered using HDBSCAN*. Other algorithms were used for experimentation, such as the K-means algorithm. However, k-means assigns each document to a cluster, and requires a k value, making it not ideal for our approach since we wanted the topics to be discovered automatically and to separate documents that were not necessarily related to each other. We experimented with HAC using Ward's method, obtaining good results, however, just as in k-means, it assigned each document to a cluster, and, with this method, it was hard to find a cut that creates clusters with documents that belong to the same topic. Further, we experimented with DBSCAN, but because of the different densities in the data it was hard to find a parametrization for each segment. Finally, HDBSCAN* was chosen to cluster the data, because it was able to detect clusters and outliers using a hierarchical approach and finding local cuts automatically.

Since HDBSCAN* can detect noise, our goal with the clustering was to reduce the number of noise points while discovering the maximum number of clusters with specific topics. More specifically, we maximized the following function:

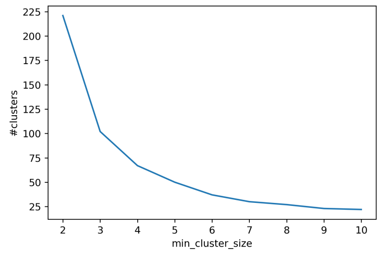
$$Z = |\xi| - \frac{|C_0|}{|D|},$$

where $|\xi|$ is the number of clusters discovered, $|C_0|$ is the number of outliers discovered and $|D|$ is the number of data points in the dataset. To satisfy these conditions, we set the parameter `min_points = 2`, making the smallest cluster possible to be composed of two documents, and the parameter `min_samples = 1`, allowing for a larger number of smaller clusters [33].

We firstly clustered the whole dataset without segmentation, finding a total of 221 clusters and 44.4%(555) outliers. We experimented with HDBSCAN* parameters to understand how they affected the clustering, since our goal was to minimize the number of outliers while maximizing clusters with unique topics. The two parameters are, `min_cluster_size` and `min_samples`. We first studied the `min_cluster_size` parameter, that defines the minimum number of data points that are required to form a cluster. We clustered the whole dataset fixing the parameter `min_samples = 1` and varying `min_cluster_size` between 2 and 10. The results for the number of clusters and percentage of outliers discovered are shown in Figure 4.2. As expected, the number of outliers increased w.r.t. the minimum number of data points required to form a cluster, while the number of clusters decreased rapidly. Since the value that minimizes the percentage of outliers is 2, we set `min_cluster_size = 2` and proceeded to investigate the next parameter by clustering the entire dataset with `min_cluster_size = 2` and varying `min_samples` between 1 and 10. Results are shown in Figure 4.3.

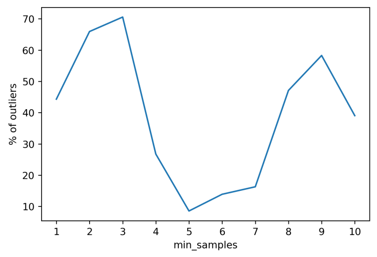


(a)

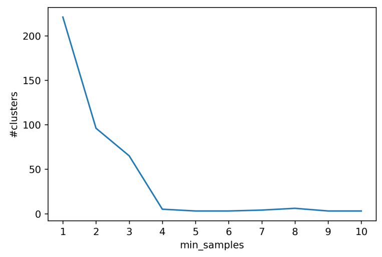


(b)

Figure 4.2: percentage of outliers w.r.t. `min_cluster_size`(a), and number of cluster w.r.t `min_cluster_size`(b)



(a)



(b)

Figure 4.3: percentage of outliers w.r.t. `min_samples`(a), and number of cluster w.r.t `min_samples`(b)

The clustering algorithm found a local minimum when `min_samples = 5`, illustrated in Figure 4.3 (a), with a small number of outliers, but also a small number of clusters. We illustrate this case in Figure 4.4 using TSNE [30] to reduce dimensionality of the data.

Henceforth, we clustered the data using `min_cluster_size = 2` and `min_samples = 1`, since these are the values that maximize the proposed func-

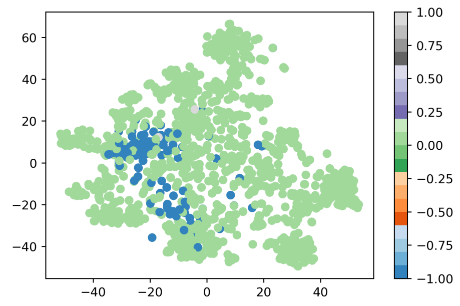


Figure 4.4: Clustering of the dataset using HDBSCAN* with `min_cluster_size = 2` and `min_samples = 5`. Data projected using TSNE.

tion Z for this specific dataset.

4.1.7 Similarity

We compared clusters in each pair of consecutive years within the analyzed period. This comparison was done in a pairwise manner using Cosine similarity as the similarity metric. Firstly, we computed the mean of each collection of tf-idf vectors that represented the documents within each cluster and computed the cosine similarity between the mean vectors. Cosine similarity returns a value between -1 and 1, where -1 means that the vector representation of the compared documents are divergent, 0 means that they are not similar and 1 means that they are very similar or equal. We used the similarity values to compute the evolution graph, creating an edge between pairs of clusters with positive similarity values.

4.1.8 Evolution Graph

We computed an evolution graph using the clusters and similarity values. A node was created for every cluster in a segment and an edge between two nodes was created for every positive weighed relationship between two clusters in consecutive segments.

This EG could be visualized in the form of a Directed Acyclic Graph (DAG). For a better visualization, we computed a word cloud of each cluster representation that was used for visualizing individual nodes. A word cloud is a popular technique where a bag of words is plotted inside a geometric shape. Words that have a larger weight are plotted larger, while words that have a lower weight are plotted smaller. Most commonly, word clouds weights are simply the raw frequencies, in this work we used the tf-idf weights. We also modified the thickness of an edge based on the similarity value. Since the graph was huge and plotting all edges made the visualization cluttered, we only plotted nodes from 2013 to 2017 and edges that had similarity values of at least 0.5. We illustrate this graph in Figure 1, in Appendix A.

4.1.9 Experiment 1 - Frequent Topic

The first question that we addressed in this dataset was: *"Is there a topic that is frequent over time?"*. This question was motivated by the initial data exploration performed and described in Section 4.1.4, where we found that topic specific words had high tf-idf values.

To address this question, we first investigated the dataset as a whole, looking at frequent words after clustering it with HDBSCAN*. We found that when we clustered the dataset without segmentation a collection of words had higher tf-idf scores than the rest of the vocabulary. The top 10 words w.r.t. tf-idf score are: *data, system, model, network, application, design, software, wireless, device and program.* Since, the tf-idf gives higher score to words that are more significative w.r.t. the data [44], we investigated further.

We used the computed Evolution Graph to analyze each cluster's top 5 words w.r.t the tf-idf score. We find that the words *"wireless"* and *"network"* appear in the top 5 words of at least one cluster in all years but 2013 (91% frequency). We also find that the remainder words (*"data", "system", "model",* *"application", "design", "software", "device" and "program"*) are mostly jargon from the computing science field, most commonly appearing in the outlier group. Our conclusion with this experiment was that the terms "wireless" and "network" were frequent w.r.t. both the whole dataset and to the discovered temporal clusters. We proceeded to investigate the evolution graphs of these two terms.

4.1.9.1 Monitoring "wireless" and "network"

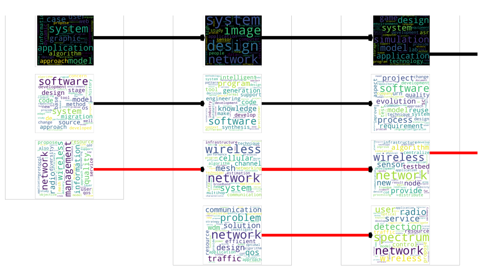
Using the evolution graph, we could monitor one or more clusters. Since we found that the terms "wireless" and "network" were representative of the dataset w.r.t. the tf-idf values, we explored the Evolution Graph searching for clusters that contained any of the two terms in its top words. We used our method described in Section 3.2.5 to compute a collection of Q-evolution graphs that contained the aforementioned terms. We filtered the EG keeping any clusters that contained any of the two terms, applying the same criteria to different timestamps. We also found that these two terms do not appear separately, thus, extracting the EG_Q using a criterion that found one of the terms also found the EG_Q of the other term. Without the evolution graph this process would be tedious, requiring extensive manual analysis of the data. Because of the huge size of the graph we did not plot the whole EG[3] in thesis, but we show two highlights in Figure 4.5.

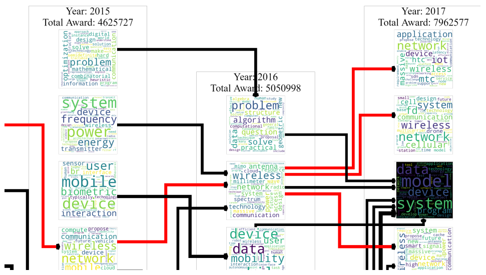
In Figure 4.5 (a) we found two concurrent topics of research in 2007 and 2008. A closer inspection of the data in those clusters showed that one topic was related to "network infrastructure", while the other topic was related to "algorithms and optimization of wireless networks".

In (b) we monitored the topics from 2015 to 2017, finding that they narrowed down in 2016 but generated several subtopics in 2017. Another inter-

[3]The full evolution graph is available at `https://tinyurl.com/yyag27wl`

esting observation is that a different field with words like "power" and "frequency", in 2015, was highly similar to "wireless network", and has "wireless network" as descendant. In 2017, when the topic of "wireless network" has several subtopics, we observed that topics like "5G network" appear, which are known to be related to both of the aforementioned topics.


(a)



(b)

Figure 4.5: Monitoring of the terms "wireless" and "network" from 2006 to 2009 (a); and, from 2015 to 2017 (b). The edges that show evolution of the topic are coloured in red.

4.1.9.2 Noise group

The research proposals that we part of the noise group were those which did not have any other proposal related to them w.r.t cosine similarity. This is true since we set the minimum size of a cluster to be two, and the clustering criterion `min_samples` = 1, which allowed HDBSCAN* to discover smaller clusters [33]. We also found that the word cloud of a group of outliers was commonly represented by general terms of Computing Science such as "system", "application", "model" and "design". This was because tf-idf weighed words that were more representative of the document, and since these documents had unique characteristics, their commonalities received higher scores.

4.1.10 Experiment 2 - Investigating Topics and Funding

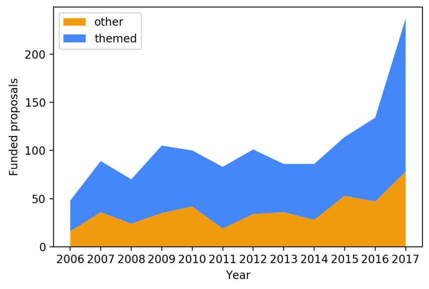
We investigated whether or not the dataset was dominated by a topic, which could help us answer questions such as: *"Is there a research topic that receives* *most of the funding provided by NSERC?"*. To answer this question, we defined that a research topic that received most of the funding would be a *dominant* topic. We define dominant as

$$has\_dominant\_topic(D) \text{ iff } \{S \in D : |S \cap \bigcup D| > \frac{|\bigcup D|}{2}\} \neq \varnothing.$$

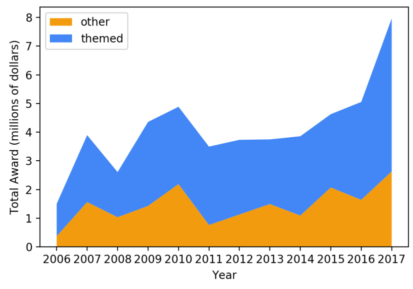
In other words, a collection of sets D contains a dominant set S if and only if there is a set $S \in D$ that contains more than half of the datapoints in the collection D. In our dataset, D is each clustering ξ and each cluster $C \in \xi$ is a set S.

Firstly, we addressed the question: *"Is there a dominant research topic in the dataset?"*. We clustered the whole dataset without segmentation and we found that 44% of the data was labeled as noise. The remaining 219 clusters shared 66% of the data, which showed that there was not a single topic that covered at least 50% of the data. We then clustered the data per year. We found that no cluster or group of outliers represented more than 44% of a segment.

Further, we investigated if the funding was mostly concentrated in a single research topic. We found that the amount of funds invested in different clusters was proportional to the number of proposals in that cluster, thus, funding also was diverse, with no large concentrations of investments in one single cluster. For example, the *"Deep Learning"* topic cluster was the most well funded in 2017. It accounted for 5.3% of the year's investments and contained 5.06% of the awarded proposals. In Figure 4.6 we illustrate the number of funded proposals (a) and the total amount awarded (b) in a given year to the noise group, labeled as "other" (orange) and clusters that had a theme, labeled as "thematic" (blue).



(a)



(b)

Figure 4.6: Number of funded proposals per year (a) and Amount of funding awarded per year, in millions of dollars (b)

4.1.11 Experiment 3 - Transition-based cluster monitoring

In this experiment we used MONIC to discover transitions in the NSERC dataset. We first computed an evolution graph using Cosine similarity and used MONIC to detect transitions. Adjusting MONIC parameters for the NSERC dataset was challenging, since small changes in τ or λ caused the number and type of discovered transitions to change. We investigated the mean similarity value and its respective deviation, illustrated in Figure 4.7. We found that the variability of the similarity value was high, and when we applied the standard values, $\tau = 0.5$ and $\lambda = 0.3$, suggested in both [48] and [41] we did not detect any survival transitions. That was because the similarity never reached a value over 0.5. Consequently, the only type of transition that could be detected was *split*, since the similarity values could sum up to higher than τ.

To investigate the applicability of MONIC to the NSERC dataset further, we focused on survival, split and merge transitions when applying MONIC to the NSERC dataset. We studied the effect of the parameters τ and λ on the transitions discovered. We first set $\tau = 0.5$ and vary λ from 0.1 to 0.5 with a step of 0.1. Table 4.1 shows the results obtained. We found that, because

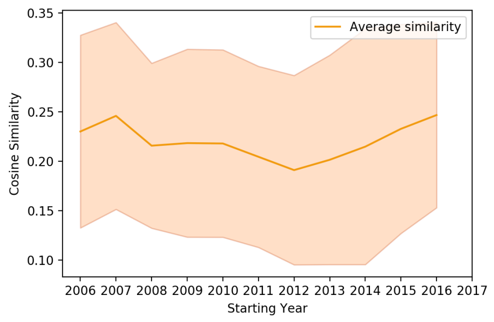


Figure 4.7: Average cosine similarity value and standard deviation.

split and merge transitions are mutually exclusive in MONIC, we most likely find one type or the other.

λ	Survival	Split	Merge
0.1	0	209	0
0.2	0	201	0
0.3	0	146	0
0.4	12	50	0
0.5	34	0	9

Table 4.1: Transitions detected by MONIC when varying λ

We explored the thresholds further, by setting $\lambda = 0.5$ and varying τ from 0.5 to 1 with a step of 0.1. Table 4.2 shows the results obtained. We found that, in the NSERC dataset, we could not discover splits and merges simultaneously. That was because the similarities between clusters were low, the variability of the similarity values was high and, split and merge are mutually exclusive, preventing both transitions to be detected if they have a common cluster.

τ	Survival	Split	Merge
0.5	12	50	0
0.6	3	59	0
0.7	0	61	0
0.8	0	61	0
0.9	0	55	0
1	0	44	0

Table 4.2: Transitions detected by MONIC when varying τ

4.1.11.1 Discovering transition-like patterns

We applied out framework to the NSERC dataset to search for patterns that were like transitions. We first segmented the graph into tuples of two consecutive timestamps, which resulted in a collection of bipartite graphs, where each graph in the collection that was a subgraph of the original evolution graph. We then labeled each node in each bipartite graph in the created collection using Equation 3.1, and then labeled each edge using Equation 3.2.

To detect a pattern that is equivalent to a survival transition, we set the minimum number of nodes of a pattern, $k_{min} = 2$ and the maximum number of nodes of a pattern, $k_{max} = 2$, since a survival pattern only involves two nodes (one at time t_i and one at time t_{i+1}). We set the minimum support $\epsilon = 1$, since a transition has to occur only once to be detected. We then executed gSpan on the collection of bipartite graphs to search for subgraphs that were equivalent to a survival transition. We found a total of 35 patterns in the dataset where 31 of those patterns were distinct from the others. We then classified each pattern w.r.t. the edge label. We found 16 patterns that were classified as "low", since the edge weight was in the range from 0 to 0.3; 11 patterns were classified as "medium", since the edge weight was in the range from 0.3 to 0.6; and, 4 patterns were classified as "high", since the edge weight was in the range from 0.6 to 1. When we raised the support value to 2, only 8 patterns were found. From those patterns 4 were distinct. We classified 2 patterns as "low", 2 patterns as "medium" and, 1 pattern as "high". Observe that, instead of giving a threshold, we classified the patterns into buckets. If we were to use MONIC with the standard value $\tau = 0.5$, patterns in the "low" bucket would not be discovered, patterns in the "medium" bucket were not very likely to be discovered, and patterns in the "high" bucket would be discovered.

To detect split and/or merge transitions we set the minimum number of nodes of a pattern, $k_{min} = 3$ and the maximum number of nodes of a pattern, $k_{max} = \infty$, since a split involves at least three nodes (one node at time t_i and at least two nodes at time t_{i+1}). Similarly, a merge transitions involves at least three nodes (at least two nodes at time t_i and one node at time t_{i+1}). If a pattern contained more than one node in both timestamps it meant that it was equivalent to simultaneous split and merge transitions. We used the same collection of bipartite graphs that were used to detected survival transitions, as described above. We stop the ex With $\epsilon = 2$ we discover only 10 patterns. The discovered patterns indicated that many of the similar patterns happened in the biennium of 2010-2011, and 2014-2015. We illustrate one of the discovered patterns in Figure 4.8, where a cluster had high similarity value to two other clusters in the subsequent year.

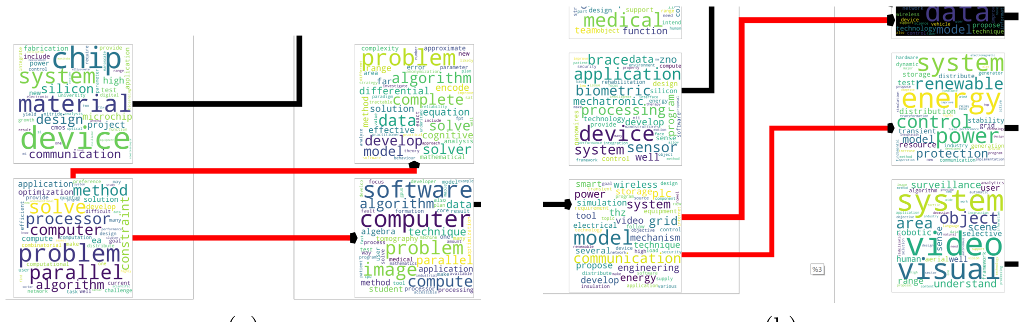


Figure 4.8: In both diagrams a cluster had high similarity value to two other clusters in the subsequent year. The pattern is detected even though the topics are different.

In addition, we investigated the NSERC dataset for larger patterns that could be discovered by our framework with support larger than one. However, because of the small size of the dataset, we were not able to find any of such patterns. We then investigated a larger dataset searching for larger patterns with frequency higher than one.

4.2 S&P 500 Dataset

The Standard & Poor's 500 or simply S&P500 is an index that is used as a benchmark in Economics. It is composed by approximately 500 stocks from both the New York Stock Exchange (NYSE) and Nasdaq Stock Market (NASDAQ). The stocks that compose the index are curated by the S&P Dow Jones Indices.

I collected the S&P500 dataset using DataStream [46], a global economics platform that provides data and fundamental information about a number of companies. The S&P500 index exists since 1923, and each stock in the index has a symbol associated with it. A symbol is an abbreviation or a unique id that refers to a given stock information, e.g., one of Google's symbols is GOOG. For the purpose of this research I used a subset of the data from 1973 to 2017, collecting weekly price information about the stocks. I did so because before 1973 many companies did not have associated symbols, making it challenging to retrieve stock prices for those companies. The final dataset was composed of 646 stocks that were part of the S&P500 index at least once during the specified time period and the dataset contained 2360 data points.

4.2.1 Data Processing

To process the dataset, we used a similar process to the one proposed by Wu et. al. [54] and used by many other authors to cluster time series data [45]. Since we accounted for different prices and stocks that joined or left the index over time, we represented a stock in a general form. Firstly, we collected the adjusted closing price, which is a monetary value of a stock that takes in account dividends and the release of new shares, allowing for a more precise pricing and avoiding abrupt price changes.

Since many companies joined or left the index, we filled the non-existent information about those companies by making their price constant w.r.t the last known price. For companies that have joined the index, we filled the previous time stamps with the first known price. For instance, a company that was created in the year 2000 at a price of $50.20 had all previous registers filled with the initial price [54].

4.2.2 Stock Representation

We normalized the values of a stock by computing the z-score of its values w.r.t. all companies that were within a segment. A z-score represents how distant a value is from the mean of a population, and is calculated as

$$z\text{-}score(x) = \frac{x - \mu}{\sigma}.$$

In other words, a z-score or standard score of a value x is x minus the mean (μ) of the population divided by the standard deviation (σ) of that population. This score provided a standardization of the price and made it independent of price fluctuations over time. The representation of a stock was the standardized vector of values of that stock w.r.t. a time period [54]. Figure 4.9 illustrates a stock representation using a vector of values within time period of four weeks.

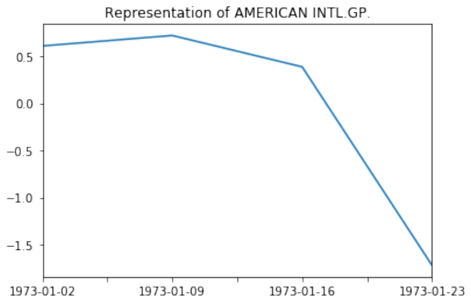


Figure 4.9: A stock was represented by its vector of prices given a segment

4.2.3 Segmentation

We experimented with segmentation of the S&P500 dataset according to three different windows: month (4 weeks), quarter (13 weeks), and year (54 weeks). These windows were defined according to established methods from the field of accounting and finance[4]. The segments were subsets of the dataset without overlap. Thus, a company C was represented by a sequence of prices $C = \langle p_0, p_1, ..., p_n \rangle$ and the cardinality of C was indicated by the length of the segment. We obtained 589, 181 and, 43 segments for 4, 13 and, 54 weeks segments, respectively.

Finally, we decided to run our experiments the dataset segmented into quarters based on two arguments: i) the portfolio of stocks in the S&P500 is updated quarterly; and, while the segmentation into years created an evolution graph of small size, weekly data generated a huge graph, that had a lot of noise and was computationally expensive to process.

4.2.4 Clustering

The stocks within each segment were clustered using HDBSCAN*. We explored other algorithms for clustering, however, because of density changes, HDBSCAN* showed better results. Our goal with clustering in this data was to find stocks that had similar price changes in the same segment. To define the minimum cluster size, we researched about Sectors and Industries[5] within the S&P500 context, which provided us with information on the different types of stock involved in the index. We found that a sector contained at least 6 stocks, therefore, we set the minimum cluster size to 6. Moreover, we set `min_samples = 1`, which allowed HDBSCAN* to discover a larger number of small clusters [33]. Euclidean Distance was the distance metric used in the

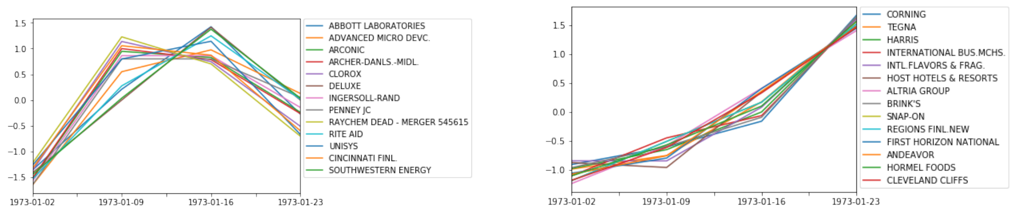
[4]https://www.irs.gov/publications/p538

[5]https://ca.spindices.com/index-family/us-equity/sector-industry

clustering process.

The *content* of a cluster was the collection of stocks and its respective prices within a segment. We could visualize a cluster by plotting all stock pricing within that cluster. Figure 4.10 illustrates two examples of cluster contents. In (a) we observe a cluster where the stock prices rose then lowered; in (b) we illustrate a cluster where stock prices rose consistently.

We used two distinct cluster representations: i) the *price representation* of a cluster was the mean vector of price of all stocks in that cluster; and, ii) the *stock representation* of a cluster was the collection of all stock symbols within that cluster. The price representation does not rely on object identity and simply on the prices of all stocks given a time period. The stock representation relies on the object identity only, since each stock had an associated symbol which was used to compute the similarity value.



(a) (b)

Figure 4.10: Cluster contents in the S&P500 dataset

4.2.5 Similarity

We used two distinct similarity functions in our experiments. The first was Jaccard Similarity, where we used the stock symbols in each cluster to find a similarity value. To compute the similarity values using the Jaccard similarity we used the stock representation. The second similarity metric used was the Cosine Similarity, where we used the price vectors of two clusters. To compute similarity values using the Cosine Similarity we used the price representation.

By computing similarity values using object identities we could infer information about stocks that perform similarly. On the other hand, by using only the prices for similarity computation we could compute a similarity value based on price vectors, not taking in account what stocks were part of a cluster.

We investigated the characteristics of both similarity metrics in terms of mean similarity and variability. When we computed the similarity values using Cosine Similarity, we found that, on average, the clusters in consecutive segments were distinct, indicated by the average value close to 0. Also, we found a large variability in the similarity value, while indicated that clusters in consecutive segments could be very different or very similar. Figure 4.11 (a) illustrates the mean and the variability of the computed Cosine Similarity.

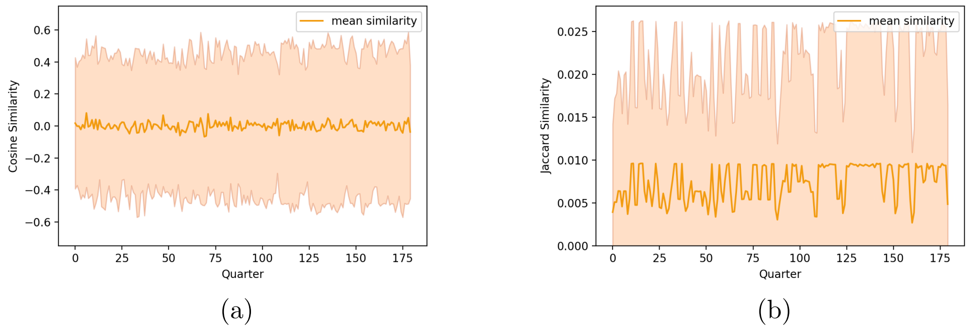


Figure 4.11: Cluster contents in the S&P500 dataset

Differently from Cosine Similarity, Jaccard Similarity only produces positive values. The values computed by the second metric were small, which indicated low similarity between clusters. Moreover, just as Cosine Similarity, the variability of the similarity values was high. We investigated the correlation of both similarity metrics by computing the Pearson correlation of their respective mean vectors. We found that the metrics were not strongly correlated, with a correlation coefficient of 0.11.

4.2.6 Evolution Graph and Visualization

Using the computed similarity values, we contructed two distinct evolution graphs, one using Jaccard Similarity values and one using Cosine Similarity values. In these EGs, we created a node for each cluster and an edge between two consecutive clusters if the similarity value between the two clusters was positive.

For the visualization of this dataset we implemented a tool that plots the EG as a DAG. Moreover, we used circle nodes for clusters and squares for noise groups. The nodes were scaled w.r.t. the amount of data points in that given node. Edges had its thickness scaled to reflect the similarity value that they represented. Because of the huge size of the EG by quarters we were not able to include it in full in this thesis[6].

4.2.7 Experiment 1 - Transition-based Pattern Detection

In this experiment we executed MONIC in the S&P500 dataset to perform transition detection. We studied its parameters and how they affect the transition detection process. We first experimented with Cosine Similarity studying both MONIC and our framework. Further, because of the low similarity values computed by Jaccard Similarity, MONIC is not capable of detecting transitions. We show that, when using Jaccard similarity, our framework can detect transition-like patterns in addition to other complex patterns.

4.2.7.1 Cosine Similarity Evolution Graph

Firstly, we experimented with MONIC in an EG computed using Cosine Similarity. We discovered that, because of the high-variability and low similarity values, we were not able to detect many transitions, except for survivals. For

[6]https://tinyurl.com/y6qceyoz

λ	Survival	Split	Merge
0.0	131	21	0
0.1	407	18	4
0.2	411	15	4
0.3	416	3	4
0.4	420	1	6
0.5	425	0	7

Table 4.3: Transitions detected by MONIC when varying λ

τ	Survival	Split	Merge
0.5	407	18	4
0.6	393	18	1
0.7	383	17	0
0.8	378	16	0
0.9	365	19	0
1	0	22	0

Table 4.4: Transitions detected by MONIC when varying τ

both similarity functions, we set $\tau = 0.5$ and varied λ from 0.0 to 0.5 using a step of 0.1. Table 4.3 shows the results obtained for Cosine Similarity. We found that MONIC detected more splits and merge transitions when $\lambda = 0.1$. Moreover, λ strongly affected the number of split transitions discovered, since the similarity values are mostly low.

Further, we investigated the effects of τ over MONIC transition detection. We set $\lambda = 0.1$ and varied τ from 0.5 to 1, with a step of 1. Since in MONIC $\tau \geq 0.5$, and because the similarity values found in the S&P500 dataset were low, the threshold change did not strongly affect the number of transitions detected.

After experimenting with MONIC, we used our framework to detect transition-like patterns without threshold restrictions. We segmented the evolution graph into a collection of bipartite graphs composed of subsequent segments. This is because are detected in two consecutive segments. We then label each bipartite graph using Equation 4.4.

$$g(e_{v,v'}) = \begin{cases} 0, \text{if } 0.0 < w(e_{v,v'}) \leq \frac{\sigma}{2} \\ 1, \text{if } 0.23 < w(e_{v,v'}) \leq \sigma \\ 2, \text{if } \sigma < w(e_{v,v'}) \end{cases} \tag{4.4}$$

In Equation 4.4 we set values that were higher than the standard deviation as *high*, represented by the value 2. Values smaller than half of the standard deviation were labeled as *low*, represented by the value 0. The remaining values were labeled as *medium*, represented by the value 1. We did not search for survival transition equivalent patterns, since the graph was huge and such patterns are abundant.

We set $k_{min} = 3$, since a split or a merge transition is composed by at least three nodes. We set $k_{max} = \infty$, since there is no limit for the number of nodes in a transition. We investigated the number of patterns discovered w.r.t. ϵ. Our framework detected more transition-like patterns than MONIC because it scans the bi-graphs for different patterns when $\epsilon = 1$. When we searched for patterns that appeared more than once we find 130 patterns that are meaningful w.r.t. ϵ. When compared to MONIC, our framework detected all transitions, with the option of filtering frequent transitions. The most frequent transition-like pattern discovered, with $\epsilon = 56$, is illustrated in Figure 4.12.

4.2.7.2 Jaccard Similarity Evolution Graph

We experimented with our framework on a Jaccard Similarity Evolution Graph. In this setting MONIC could not detect any transitions because of the low similarity values. On the other hand, our framework detects transition-like patterns.

We set $k_{min} = 3$, since a split or a merge transition are composed of at least three nodes. We set $k_{max} = \infty$, since there is no limit for the number

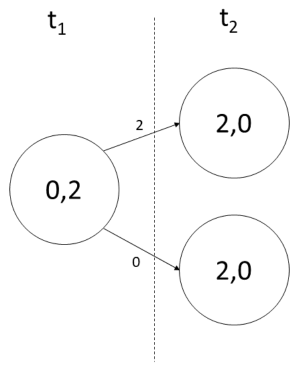


Figure 4.12: Most frequent transition-like pattern detected by our framework when using Cosine Similarity

of nodes in a transition. We investigated the number of patterns discovered w.r.t. ϵ.

When we investigate for patterns that appeared more than once ($\epsilon = 2$), we find 68 patterns that are meaningful w.r.t. ϵ. The most frequent transition-like pattern discovered, with $\epsilon = 56$, is illustrated in Figure 4.13. This pattern is composed of three nodes and indicate that there is a strong relationship between the node in t_1 and at least one of the nodes in t_2. Also, there exists a weak relationship between the node in t_1 and the other node in t_2. Both nodes in t_2 indicate that there is another node influencing them, indicated by the labeling $2, 0$. The Cosine Similarity function compares the vector of the two nodes, which means that both clusters in t_2 have vectors that are somewhat similar to the vector of the node in t_1 but significantly different among themselves.

4.2.8 Experiment 2 - Transition-free Pattern Detection

To explore the dataset further with our framework, we searched the evolution graph by segmenting it into a collection of k-partite graphs where $k > 2$. We set $k_{min} = n + 1$, $k_{max} = \infty$ and $\epsilon = 3$. We chose these values because we had a partition of n segments, by setting $k_{min} = n + 1$ we guaranteed that the

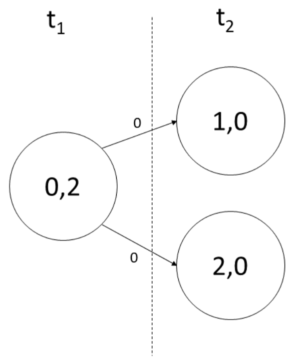


Figure 4.13: Most frequent transition-like pattern detected by our framework when using Jaccard Similarity

pattern was not a $\lambda - chain$ of n nodes, and at least one of the nodes had a relationship to two other nodes. We illustrate the most frequent pattern for $k = 3$ in Figure 4.14. The Figure shows a patterns composed of three nodes where the similarity values in both edges are low but sufficient to establish a relationship between the nodes. Since the similarity function compares the set of stock symbols contained in each node, this pattern indicates that, often, a group of nodes of arbitrary size is similar to two other groups of companies. Moreover, the lower node in the figure indicates two inputs, one of which is not part of the pattern, meaning that there is some external influence from other clusters, but less frequently than this three node pattern.

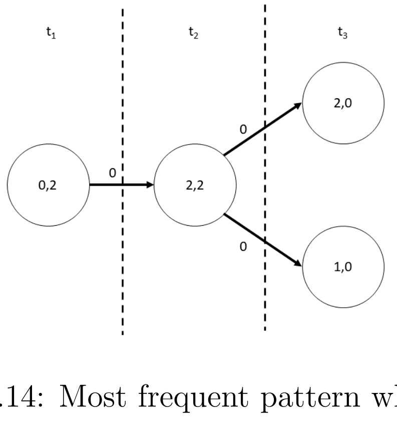


Figure 4.14: Most frequent pattern when $\epsilon = 3$

To understand the meaning of these patterns we investigated the representation of the clusters and the changes that were occurring in the prices. We find a general trend that shows subtle changes in the prices, maintaining the vectors similar enough so the similarity value is positive. That means that our framework found mostly patterns that indicate small price changes in stocks, not accounting for significant changes.

4.3 Discussion

In this section we have experimented with our proposed framework using two real-world datasets: NSERC and S&P500. We demonstrated that, when detecting transitions, our framework could detect transitions detected by the state-of-the-art and transitions that would not be detected because of different similarity value ranges.

We showed that our framework can track the evolution of a cluster, tracking a set of words in the NSERC dataset. Moreover, we showed that our framework could aid on extracting insights from temporal datasets.

When we explored a dataset where the similarity values were low, our framework could still find patterns, while a traditional approach could not. We also showed that our framework could discover patterns that are more complex than transitions. These patterns could not be detected by a traditional approach because of low similarity values, or significant variability of the similarity values over time. This is because the state-of-the-art looks for one-step patterns, while our framework mines a graph for multiple-step patterns. Moreover, our framework is not limited by a threshold value for the similarities, rather using bucketing of that similarity to label the patterns, search for the most frequent ones. Finally, because of the bucketing, our framework can adapt to variability in the similarity values, which traditional approaches cannot.

Chapter 5

Conclusion

In this thesis we addressed the problem of Cluster Monitoring. The problem consists in, given a dataset, trace the evolution of clusters over time so that insights about the changes occurring in the data over time can be extracted. We tackled this problem by proposing a pattern-based framework and extracting patterns that are more complex than transitions.

We experimented with our pattern-based framework in two real-world datasets: the NSERC dataset the S&P500 dataset. We show that our framework is capable of detecting both transitions and complex patterns. Moreover, our framework is flexible in terms of accepting different types of similarity functions, clustering algorithms, and both labeled and unlabeled data.

5.1 Future Work

There are many avenues that can be explored to improve this work. Our framework can be used to detect frequent temporal patterns and summarize an evolution graph. Moreover, our framework can be used to compress a graph not only in terms of the survival transitions but in terms of complex patterns.

In the field of community mining, our framework can be applied to detect changes in both the entity level and the community level. By detecting frequent changes in the community, our framework can be further extended to predict future changes in the community structure. Finally, in the field of Economics, an avenue is to explore other similarity functions, or the use of the negative values computed by the Cosine Similarity function to investigate market shifts.

References

[1] C. C. Aggarwal, "A framework for diagnosing changes in evolving data streams," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ACM, 2003, pp. 575–586. 10, 16

[2] ——, "On change diagnosis in evolving data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 5, pp. 587–600, 2005. 9, 10, 16

[3] C. C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," in *Mining text data*, Springer, 2012, pp. 77–128. 52

[4] ——, *Mining text data*. Springer Science & Business Media, 2012. 52

[5] F. Alkemade and C. Castaldi, "Strategies for the diffusion of innovations on social networks," *Computational Economics*, vol. 25, no. 1-2, pp. 3–23, 2005. 20

[6] S. Asur, S. Parthasarathy, and D. Ucar, "An event-based framework for characterizing the evolutionary behavior of interaction graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 3, no. 4, p. 16, 2009. 16, 19, 21, 23

[7] S. Baron, M. Spiliopoulou, and O. Günther, "Efficient monitoring of patterns in data mining environments," in *East European Conference on Advances in Databases and Information Systems*, Springer, 2003, pp. 253–265. 9

[8] I. Bartolini, P. Ciaccia, I. Ntoutsi, M. Patella, and Y. Theodoridis, "A unified and flexible framework for comparing simple and complex patterns," in *European Conference on Principles of Data Mining and Knowledge Discovery*, Springer, 2004, pp. 496–499. 10

[9] M. Böttcher, F. Höppner, and M. Spiliopoulou, "On exploiting the power of time in data mining," *ACM SIGKDD Explorations Newsletter*, vol. 10, no. 2, pp. 3–11, 2008. 1

[10] S. Brohee and J. Van Helden, "Evaluation of clustering algorithms for protein-protein interaction networks," *BMC bioinformatics*, vol. 7, no. 1, p. 488, 2006. 8

[11] K. Chen and L. Liu, “Detecting the change of clustering structure in categorical data streams,” in *Proceedings of the 2006 SIAM International Conference on Data Mining*, SIAM, 2006, pp. 504–508. 10, 16

[12] D. G. Corneil, H. Lerchs, and L. S. Burlingham, “Complement reducible graphs,” *Discrete Applied Mathematics*, vol. 3, no. 3, pp. 163–174, 1981. 39

[13] R. Cowan and N. Jonard, “Network structure and the diffusion of knowledge,” *Journal of economic Dynamics and Control*, vol. 28, no. 8, pp. 1557–1575, 2004. 20

[14] K. Cup, “Dataset,” *available at the following website http://kdd. ics. uci. edu/databases/kddcup99/kddcup99. html*, vol. 72, 1999. 15

[15] G. Dong, J. Han, L. V. Lakshmanan, J. Pei, H. Wang, and P. S. Yu, “Online mining of changes from data streams: Research problems and preliminary results,” in *Proceedings of the 2003 ACM SIGMOD Workshop on Management and Processing of Data Streams*, 2003, pp. 739–747. 1

[16] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, “Graphviz—open source graph drawing tools,” in *International Symposium on Graph Drawing*, Springer, 2001, pp. 483–484. 50, 51

[17] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.,” in *Kdd*, vol. v. 96, n34, 1996, pp. 226–231. 7

[18] T. Falkowski, J. Bartelheimer, and M. Spiliopoulou, “Mining and visualizing the evolution of subgroups in social networks,” in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, IEEE Computer Society, 2006, pp. 52–58. 11, 16

[19] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936. 6

[20] V. Ganti, J. Gehrke, and R. Ramakrishnan, “A framework for measuring changes in data characteristics,” in *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ACM, 1999, pp. 126–137. 9

[21] L. Geng and H. J. Hamilton, “Interestingness measures for data mining: A survey,” *ACM Computing Surveys (CSUR)*, vol. 38, no. 3, p. 9, 2006. 42

[22] J. A. Hartigan, “Clustering algorithms,” *Wiley*, 1975. 4

[23] A. Inokuchi, T. Washio, and H. Motoda, “An apriori-based algorithm for mining frequent substructures from graph data,” in *European conference on principles of data mining and knowledge discovery*, Springer, 2000, pp. 13–23. 42

[24] C. Jiang, F. Coenen, and M. Zito, “A survey of frequent subgraph mining algorithms,” *The Knowledge Engineering Review*, vol. 28, no. 1, pp. 75–105, 2013. 41, 42

[25] R. Jin, C. Wang, D. Polshakov, S. Parthasarathy, and G. Agrawal, "Discovering frequent topological structures from graph datasets," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, ACM, 2005, pp. 606–611. 39

[26] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in *Proceedings 2001 IEEE International Conference on Data Mining*, IEEE, 2001, pp. 313–320. 39

[27] ——, "Finding frequent patterns in a large sparse graph," *Data mining and knowledge discovery*, vol. 11, no. 3, pp. 243–271, 2005. 39

[28] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, "Facetnet: A framework for analyzing communities and their evolutions in dynamic networks," in *Proceedings of the 17th international conference on World Wide Web*, ACM, 2008, pp. 685–694. 21, 23

[29] E. Loper and S. Bird, "Nltk: The natural language toolkit," *arXiv preprint cs/0205028*, 2002. 51

[30] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008. 57

[31] P. Macnaughton-Smith, W. Williams, M. Dale, and L. Mockett, "Dissimilarity analysis: A new technique of hierarchical sub-division," *Nature*, vol. 202, no. 4936, p. 1034, 1964. 6

[32] K. McGarry, "A survey of interestingness measures for knowledge discovery," *The knowledge engineering review*, vol. 20, no. 1, pp. 39–61, 2005. 42

[33] L. McInnes, J. Healy, and S. Astels, "Hdbscan: Hierarchical density based clustering," *The Journal of Open Source Software*, vol. 2, no. 11, p. 205, 2017. 50, 51, 56, 61, 69

[34] T. Müller, R. Cotterell, A. Fraser, and H. Schütze, "Joint lemmatization and morphological tagging with lemming," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 2268–2274. 54

[35] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, 2002, pp. 849–856. 7

[36] E. Ntoutsi, M. Spiliopoulou, and Y. Theodoridis, "Fingerprint: Summarizing cluster evolution in dynamic environments," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 8, no. 3, pp. 27–44, 2012. 18, 23

[37] I. Ntoutsi, M. Spiliopoulou, and Y. Theodoridis, "Tracing cluster transitions for different cluster types.," *Control & Cybernetics*, vol. 38, no. 1, 2009. 14

[38] ——, "Summarizing cluster evolution in dynamic environments," in *International Conference on Computational Science and Its Applications*, Springer, 2011, pp. 562–577. 18

[39] M. Oliveira and J. Gama, "Bipartite graphs for monitoring clusters transitions," in *International Symposium on Intelligent Data Analysis*, Springer, 2010, pp. 114–124. 15, 16, 23

[40] ——, "Understanding clusters evolution," in *Workshop on Ubiquitous Data Mining*, vol. 500, 19, 2010, pp. 16–20. 16

[41] M. Oliveira and J. Gama, "A framework to monitor clusters evolution applied to economy and finance problems," *Intelligent Data Analysis*, vol. 16, no. 1, pp. 93–111, 2012. 16, 17, 23, 48, 63

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 50, 51

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011. 51

[44] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, Piscataway, NJ, vol. 242, 2003, pp. 133–142. 54, 59

[45] S. Rani and G. Sikka, "Recent techniques of clustering of time series data: A survey," *International Journal of Computer Applications*, vol. 52, no. 15, 2012. 67

[46] T. Reuters, *Datastream*. 2010. 67

[47] P. J. Rousseeuw and L. Kaufman, "Finding groups in data," *Series in Probability & Mathematical Statistics - 1990 - 34(1)*, pp. 111–112, 1990. 6

[48] M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult, "Monic: Modeling and monitoring cluster transitions," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2006, pp. 706–711. 11, 16, 23, 48, 63

[49] M. Takaffoli, F. Sangi, J. Fagnan, and O. R. Zaiane, "Community evolution mining in dynamic social networks," *Procedia-Social and Behavioral Sciences*, vol. 22, pp. 49–58, 2011. 21

[50] ——, "Modec—modeling and detecting evolutions of communities," in *Fifth international AAAI conference on weblogs and social media*, 2011. 21, 23, 25

[51] S. M. Van Dongen, "Graph clustering by flow simulation," PhD thesis, 2000. 8, 9

[52] U. Von Luxburg, “A tutorial on spectral clustering,” *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007. 7

[53] D. S. Wilks, “Cluster analysis,” in *International geophysics*, vol. 100, Elsevier, 2011, pp. 603–616. 6

[54] J. Wu, A. Denton, O. Elariss, and D. Xu, “Mining for core patterns in stock market data,” in *2009 IEEE International Conference on Data Mining Workshops*, IEEE, 2009, pp. 558–563. 67, 68

[55] X. Yan and J. Han, “Gspan: Graph-based substructure pattern mining,” in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, IEEE, 2002, pp. 721–724. 39, 53

[56] H. Yang, S. Parthasarathy, and S. Mehta, “A generalized framework for mining spatio-temporal patterns in scientific data,” in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, ACM, 2005, pp. 716–721. 11, 16

Appendix A

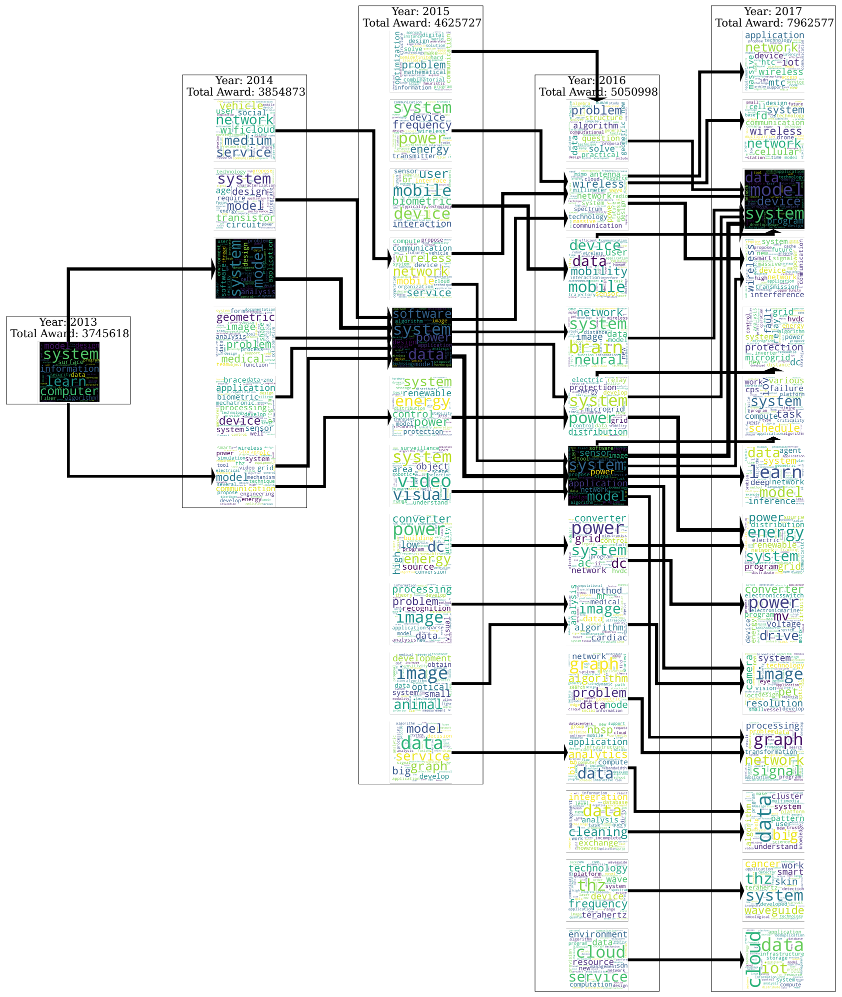


Figure 1: Evolution Graph of the NSERC Dataset from 2013 to 2017 and edges with similarity values of at least 0.5. A black word cloud represents an outlier group.