CO-CLUSTERING SIGNED 3-PARTITE GRAPHS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

SEFA ŞAHİN KOÇ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2017

Approval of the thesis:

CO-CLUSTERING SIGNED 3-PARTITE GRAPHS

submitted by **SEFA ŞAHİN KOÇ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Prof. Dr. İsmail Hakkı Toroslu
Supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Göktürk Üçoluk
Computer Engineering Department, METU _____

Prof. Dr. İsmail Hakkı Toroslu
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Sinan Kalkan
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Pınar Karagöz
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Osman Abul
Computer Engineering Department, TOBB ETÜ _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SEFA ŞAHİN KOÇ

Signature :

ABSTRACT

CO-CLUSTERING SIGNED 3-PARTITE GRAPHS

Koç, Sefa Şahin
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. İsmail Hakkı Toroslu

February 2017, 55 pages

Real world data is complex and multi-related among itself. Considering a social media, multiple users can interact with same item such as commenting, liking etc. Data composed of these actions contains many nodes from different types (user, item, sentiment). Therefore, clustering nodes with same type will not be sufficient to analyze it. It will ignore relations between nodes from different types. Such data should be dealt with heterogeneous multi-partite clustering methods. Thus, clustering does not ignore relations among different types. At the end, heterogeneous clusters are found, which are effective to represent inter-partition relations as well as intra-partition ones. To exemplify, from a complex big relations of <user,keyword,issue>, clusters may be extracted such that they contains users who uses similar sentiments to address same issues. I present a new algorithm, called STRICLUSTER, which evaluates heterogeneous data which contains relations of three different types. Each relation is called an hyperedge where each links three nodes from distinct types. Moreover, hyperedges carry a sentiment, which is either positive or negative. The algorithm finds tripartite

clusters which express high positivity. Overlap of hyperedges among clusters are not allowed while a node can be part of many clusters. Furthermore, our algorithm handles negative property and sparseness of hyperedges while discovering tripartite clusters of hyperedges with positive properties. I will show its effectiveness via experiments and results. Experiments are performed on both synthetic and real-world data.

Keywords: data mining, clustering, multi-partite, graph, twitter, sentiment analysis

ÖZ

3 KUTUPLU İFADELİ GRAFİK AĞLARININ GRUPLANMASI

Koç, Sefa Şahin
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. İsmail Hakkı Toroslu

Şubat 2017 , 55 sayfa

Gerçek veri, karmaşık ve kendi içirisinde çok ilişkilidir. Bu yüzden, türdeş kümeleme metotları bu verileri incelemek için yeterli desteği sağlamaz. Bu tarz kümeleme, farklı tür veri nesneleri arasındaki bağlantıları göz ardı eder. Buna dayanarak, gerçek verinin aynı türden olmayan nesneleri aynı kümeye koyabilecek yöntemler ile analiz edilmesi gerekir. Bu noktada, ihtiyaca cevap veren yöntem, k-kutuplu kümelemedir. Bu yöntem, farklı türden nesneler arasındaki bağlantıları değerlendirerek karışık kümeleme yapma yeteneğine sahiptir. Bulunan kümeler, hem aynı türdeki hem de farklı türdeki nesnelerin aralarındaki ilişkileri göstermek açısından etkilidir. Örneklemek gerekirse, <kişi,konu,kelime> üçlülerinden oluşan büyük ve karmaşık bir veriden, aynı konu hakkında benzer kelimeleri kullanarak olumlu ifadelerini belirten kişiler bulunabilir. Bu tezde, farklı cinsten nesneler içeren kümeler bularak bu tarz analizler yapılmasını sağlayacak STRICLUSTER algoritmasını anlatacağım. Bu algoritma 3 farklı tür ve aralarındaki üçlü ilişkilerden oluşan veri üzerinden koşar. Her bir ilişki, her bir

türden olmak üzere tam olarak 3 nesne arasındadır ve olumlu ya da olumsuz bir ifade taşır. Bu algoritma, olumlu ifadelerin yoğunlukta olduğu 3-kutuplu kümeler bulur. Bir nesne birden fazla küme tarafından içerilebilir ya da hiçbir kümeye ait olmayabilir. Bu kümeleri bulurken bir takım sezgisel yöntemler kullanır. Ek olarak, algoritmamız olumsuz ifade taşıyan ilişkileri ve boşlukları etkili şekilde kontrol altında tutar. Algoritmanın 3-kutuplu kümeleme işlemini tutarlı ve etkili bir şekilde gerçekleştirdiğini, yapay ve gerçek veri üzerinde koşulan testlerle göstereceğim. Bu testlerden elde edilen ölçümler ve grafikler, algoritmanın etkinliğini destekleyecektir.

Anahtar Kelimeler: veri madenciliği, gruplama, çok-kutuplu, twitter, fikir analizi

To my family and homeland

ACKNOWLEDGMENTS

I would like to thank my supervisor Prof. İsmail Hakkı Toroslu for his support and supervision.

I also would like to thank my current employer ASELSAN for supporting my academic research financially.

I would like to thank my previous employer TUBİTAK which supports me to do master education during my employment.

I also would like to thank Prof. Hasan Davulcu from ASU for his guidance, further, his Phd. student Mert Özer for providing data for experiments.

Finally, I would like to thank my family for their patience, understanding and support throughout whole process.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ASELSAN	Askeri Elektronik Sanayi
ASU	Arizona State University
EU	European Union
LibDems	Liberal Democrats
NHS	National Health Service
SNP	Scottish National Party
TUBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
UK	United Kingdom
UKIP	United Kingdom Independence Party

CHAPTER 1

INTRODUCTION

When data is represented on a virtual platform, it is desired to protect all information such as meta-data, relations, changes etc. Some types of data are suitable to store in a simple table. For some, these homogeneous structures are not sufficient, though. Generally, data gathered from real world systems are complex enough that requires more relational structures like graphs, also called networks [28]. The systems like a social media preserves many relations among various types of nodes. Such data with many relations can be drawn as a heterogeneous graph. It is called heterogeneous because its vertices are from different types. It is quite effective for representing not only users but also their staff and relations with them. As this approach is a quite good way of preserving data, the size of graph and its connectivity are so interesting for computer scientists. Analyzing user actions to understand minds under different circumstances is an exciting research area. This is one of the main reasons why social network analysis is quite popular.

For a simple data structure, its analysis is relatively easy. Let us consider a table. Data retrieval with desired properties can be performed by a single query which can be written in a few lines. When it comes to analyzing a homogeneous graph, more sophisticated algorithms are needed such as Dijkstra's shortest path [11], rather than a simple query. If the structure and type of analysis becomes more complicated, the applied algorithm needs an enhancement. One of the good examples to this is analyzing k-partite graphs [4]. k-partite graphs are commonly used to store relations among partite groups where each group refers

to a type. Nodes inside same partite groups are not connected among themselves. To illustrate this characteristic, <user,photo,comment> triple can be considered. Two photos are not directly connected. But, it is indirectly linked via a user who writes comments on both.

Main purpose of using k-partite graphs is generating clusters, which preserves relations among nodes from different types. Therefore, more specific analysis can be done about nodes rather than only about their similarity. A partite group is a sub-unit in the underlying cluster where it is closely related with other partite ones. Such structure will provide insight about node characteristics and composite information about the cluster. Finally, the acquisitions are more relative to other nodes. For example, by clustering pictures which share same tags, group of nature or sport pictures can be found. On the other hand, k-partite clustering can give users who like nature photos in a cluster.

In partitioning and hierarchical clustering methods, we call traditional clustering, generally homogeneous graphs are focused. In these graphs, nodes are from same type. Homogeneous clustering methods aim to find clusters of the nodes, which are highly connected among themselves but loosely connected with outside [7, 13, 16]. However, real world systems are generally more complex and includes various types of nodes (heterogeneous). Trying to cluster such systems homogeneously will cause to ignore connectivity between different types. This leads to lack of valuable information. For example, for the social network, Instagram[1], a user can upload photos and tag them with keywords. Moreover, they can follow other users as they can be followed inside that social network. A way of grouping users can be obtained using connectivity among them. Group of friends can be discovered from such clusters. However, in order to find users with same interests or same perception on same photos can be discovered by evaluating users with photos and tags. This is where k-partite clustering becomes significant.

[1] https://www.instagram.com/

1.1 Problem Definition

Considering the wide-range of applications of multi-partite clustering and great interest to this approach, we address a special version of it. The aim is to find k-partite clusters where k is constant and equal to 3. Respectively, the addressed data is a hypergraph which is composed of hyperedges which link nodes from three different types. This hypergraph can also be represented as a 3-D matrix. A hyperedge represents a connection among three nodes from different types. In input, each hyperedge carries a label, either a positive or negative. This represents the attitude of hyperedge relation. For example, a user tags a resource and this tagging represents either positive or negative opinion.

By considering this input data, we aim to find tripartite clusters with maximal size, which are mainly composed of positively labeled hyperedges. Negative ones and sparseness can be tolerated at some level. To the best our knowledge, this is a new problem in this literature.

This problem can be adapted to many systems. However, especially data gathered from social platforms are suitable. Actions can be easily represented as relations of many nodes, such as user, photo, comment, topic etc. Moreover, the attitude of each action can be more apparent. The words or action itself such as liking will point out either positive or negative feeling. The obtained clusters hold actions with similar sentiments on same objects. They can be analyzed in many ways. For example, objects about which humans express positive sentiments at particular states can be discovered. On this perspective, the results could be promising to find user groups of similar interests.

1.2 Solution

To address the problem, we will introduce a new algorithm, named as STRICLUSTER. It is applied on heterogeneous data composed of hyperedges with three distinct types of nodes and a label, either positive or negative. The algorithm finds tripartite clusters which contains mainly positive hyperedges. Fur-

thermore, it has the following properties: 1) A minimum threshold for density ratio of positively signed hyperedge over all possible hyperedges inside a cluster is defined, and, it must be satisfied by clusters. 2) A greedy approach is used in order to trim the hyperedges from tri-clusters with negative signs to increase the positive density ratio of the cluster. 3) In order to prevent constructing very small clusters, both negative signed hyperedges and triples with no connections are also allowed as long as they satisfy user defined density threshold constraints. 4) Clusters are not allowed to have overlaps in terms of hyperedges. A simple heuristic is used to mark hyperedges in order to prevent hyperedge overlaps among clusters, and fast termination of the algorithm while searching potentially maximal clusters. 5) A node can be part of one or more clusters as it can be idle. In this manner, node overlap is allowed due to positive connectivity of a node. Weak nodes or nodes with high negative connectivity are likely to remain out of clusters. 6) The effectiveness of our approach is shown using a coverage-based metric.

1.3 Outline of the Thesis

The rest of the thesis is organized as follows. Chapter 2 contains literature survey about previous researches in this area. There is brief information about graphs and hypergraphs. They are related because the input data can be represented as a hypergraph. Each link between nodes refers to an hyperedge. Additionally, homogeneous and heterogeneous clustering is exemplified. Then, the main focus area of this thesis, multi-partite clustering is explained. The example solutions on this area are given. The reason why they do not provide a solution to the problem are discussed. Finally, the differences between our solution and existing solutions are examined.

Chapter 3 starts with introducing our solution, STriCluster algorithm. Then, the algorithm is deeply explained. Heuristic approaches are given. The mainstay of these approaches is remarked.

Chapter 4 contains the experiments of STriCluster algorithm. The experi-

ments are performed on both synthetic and real data. Data acquisition for them is explained. Finally, interpretations on the test results are given.

Chapter 5 concludes the thesis with a brief summary of our solution. It also includes future works which can be added on the solution.

CHAPTER 2

LITERATURE SURVEY

Multi-partite clustering is a method to find clusters inside a graph which consists of various type of nodes. Clusters are coalescence of partitions, where each partition is composed of same type of nodes. The simplest version is bi-partite clustering which results clusters of only two partite groups. It is more common research area compared to k-partite clustering where $k > 2$. Our problem addresses 3-partite hypergraphs which consists of hyperedges which contains exactly three nodes with different types. These hypergraphs can be representable in a 3D matrix, where each entry refers to an hyperedge. Similar manner, matrix is a common structure to represent graphs. The reason is not only that matrix construction is easy on memory, but also that operations can be applied on graphs via matrix structure more easily. For better understanding, I will explaining terminologies and techniques of multi-partite clustering.

In this section, first I will explain graphs. As a representation of complex data, graphs play significant role on clustering operations.

Secondly, I will give information about clustering methodology. I will mention previous research done about it.

Finally, I will explain works related to multi-partite clustering methodology on literature, especially about tripartite clustering. Depending on their approach styles and results, I will discuss in what senses our solution is different.

2.1 Graphs

This section is written based on Bondy's book [3].

A graph consists of three main parts $(V(G), E(G), \psi_G)$, where $V(G)$ is the set of vertices, $E(G)$ is the set of edges and ψ_G is the incident function to bind two vertices with an edge. For example, for $\{u, v\} \in V(G), e \in E()$, if $\psi_G(e) = uv$, then it is said that e connects u and v.

To exemplify the definition, let $G = (V(G), E(G), \psi_G)$ where

$$\begin{aligned} V(G) &= \{u, v, x, w\} \\ E(G) &= \{a, b, c, d, e\}, \end{aligned} \tag{2.1}$$

$$\begin{aligned} &\psi_G(a) = uv \quad \psi_G(b) = vx \quad \psi_G(b) = xw \\ &\psi_G(d) = uw \quad \psi_G(e) = vw. \end{aligned} \tag{2.2}$$

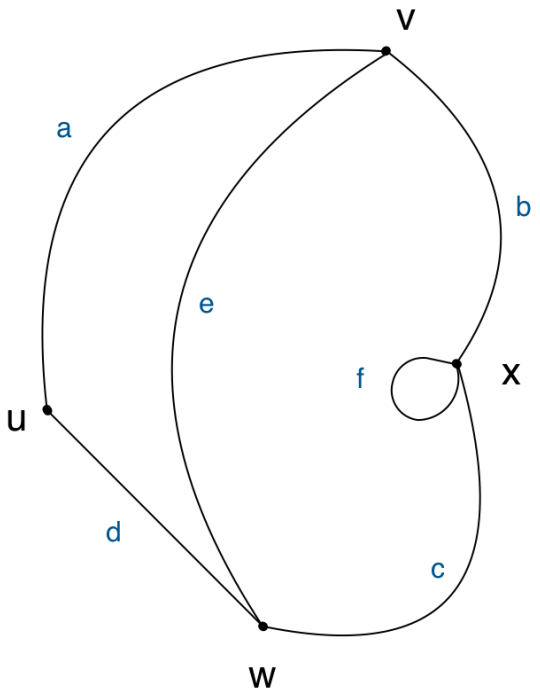


Figure 2.1: Representation of Graph G

Identical Graphs: Let G and H be graphs. They are identical if $V(G) = V(H)$ and $E(G) = E(G)$ and $\psi_G = \psi_H$. If two graphs are identical, they can be

	a	b	c	d	e	f
u	1	0	0	1	0	0
v	1	1	0	0	1	0
x	0	1	1	0	0	2
w	0	0	1	1	1	0

(a) $M(G)$

	u	v	x	w
u	0	1	0	1
v	1	0	1	1
x	0	1	1	1
w	1	1	1	0

(b) $A(G)$

Figure 2.2: Incidence and Adjacency Matrices of Graph G in Figure 2.1

represented on same diagram. However, having same diagram does not require that these two graphs are identical.

Complete Graphs: Let G be a graph. If each pair of distinct vertices in G is linked by an edge, then G is a complete graph.

Incidence Matrix: Incidence matrix of graph G, denoted by $M(G)$ is a $v \times e$ matrix whose entries m_{ij} keeps the number of times that v_i and e_j are incident, which is either 0, 1 or 2 (Figure 2.2a).

Adjacency Matrix: Adjacency matrix of graph G, denoted by $A(G)$ is $v \times v$ matrix whose entries m_{ij} keeps the number of edges by which v_i and v_j are connected (Figure 2.2b).

Degree Matrices: The degree of vertex v in G, denoted by $d_G(v)$ refers to number of edges which are incident with v. Loop edges, whose endpoints are on same vertices count twice.

Degree matrix of graph G, denoted by $D(G)$ is $v \times v$ diagonal matrix whose entries m_{ii} refers to $d_G(v_i)$ (Figure 2.3). $m_{ij} = 0$ if $i \neq j$.

	u	v	x	w
u	2	0	0	0
v	0	3	0	0
x	0	0	4	0
w	0	0	0	3

Figure 2.3: Degree Matrix of G

Paths and Connection: A finite sequence $W = v_0e_1v_1...e_kv_k$ is composed of

an ordered vertices together with edges which connect vertices on this order. All vertices from v_0 to v_k are in $V(G)$, and edges e_1 to e_k are in $E(G)$. Then, W is called a *walk* in graph G. It is also denoted as (v_0, v_k)-walk. v_0 as a starting point is called *origin*, and v_k is called *terminus* in the same manner.

A special version of walk is called trail if all edges in W are distinct. Additionally, if all vertices are also distinct, then W is called a path. For illustration, $udwevaudwcx$ is a walk in graph on Figure 2.1. On same graph, $vaudwevbx$ is a trail. $wcxbv$ is a path.

A walk whose origin and terminus is same vertex with positive length is a closed walk. A closed trail is a trail whose internal vertices are different from the origin vertex. A closed trail whose internal vertices are distinct is a *cycle*. A cycle is named depending on its length. A k-cycle is a cycle with length k. For example, when k is 3, it is called 3-cycle, often called triangle. If k is even, then k-cycle is called even. Similar, it is applied for odd. An illustration to closed trail on graph G in Figure 2.1 is $vbxfxcwev$. On the same graph, $vewduav$ is a cycle.

A graph is bipartite if it does not contain any odd cycle.

Vertex Cut: A vertex cut of connected graph G is a set of vertices V' whose removal from G makes G disconnected. k-vertex cut is cut of k number of vertices. Connectivity of G, denoted by $\kappa(G)$ is defined as size of minimal vertex cut. There is no vertex cut for complete graphs. If there is at least one pair of vertices which are not adjacent, then $\kappa(G)$ is k of minimum vertex cut in G. If there is no such vertex cut, then $\kappa(G)$ is defined as $v - 1$. Therefore, for trivial or already disconnected graphs, $\kappa(G) = 0$. if $\kappa(G) \geq k$, then it is said that G is k-connected. This brings the corollary that all non trivial connected graphs are 1-connected.

2.2 Hypergraph

A hypergraph [2] $H = (V, E = (e_i)_{i \in I})$ is the family of $(e_i)_{i \in I}$ of subsets of V, such that

$$e_i \neq \emptyset \quad (i = 1, 2, ..., m), \tag{2.3}$$

$$\bigcup_{i=1}^{m} e_i = V. \tag{2.4}$$

V is the set of vertices $\{v_1, v_2, ..., v_n\}$ and $\{e_1, e_2, ..., e_m\}$ is the set of edges of hypergraph. e_i is also called as hyperedge. Number of vertices which e_i connects is defined as cardinality of e_i. This terminology is shortly represented as $|e_i|$.

A hypergraph of $E = \{e_1, e_2, ..., e_m\}$ is simple hypergraph (or "Sperner family") such that

$$e_i \subset e_j \Rightarrow i = j. \tag{2.5}$$

A simple graph is a simple hypergraph such that

$$|e_i| = 2 \quad (i = 1, 2, ..., m). \tag{2.6}$$

A hypergraph H can be visualized as a set of points which represent vertices. Edges are drawn depending on their cardinality. If $|e_i|$ is 1, then e_i will be a circle on the node. If $|e_i|$ is 2, then two vertices will be connected by line. If $|e_i|$ is greater than 2, the edge will be a closed curve which contains all vertices (Figure 2.4).

A hypergraph H can be also defined with an incident matrix $A = ((a_i^j))$. Columns are representing $e_1, e_2, ..., e_m$ and rows are $v_1, v_2, ..., v_n$. a_i^j is 0 if $v_i \notin e_j$, a_i^j is 1 if $v_i \in e_j$ (Figure 2.5).

2.3 Clustering

Grouping a set of objects depending on their similarities as classes is called clustering. A cluster contains objects which are similar among ones inside class, but less similar to objects of different classes. Clustering is a widely used method

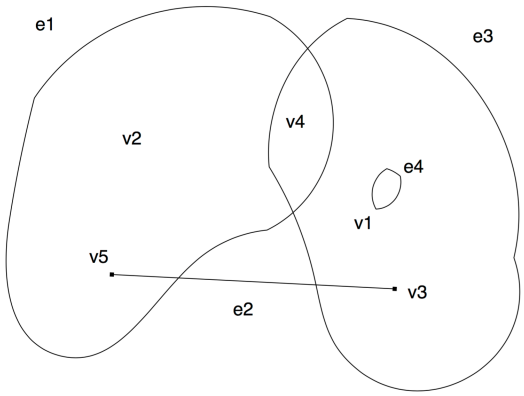


Figure 2.4: Representation of a Hypergraph

	e1	e2	e3	e4
v1	0	0	1	1
v2	1	0	0	0
v3	0	1	1	0
v4	1	0	1	0
v5	1	1	0	0

Figure 2.5: Incident Matrix of the Hypergraph in Figure 2.4

in many areas, such as marketing, social analysis, machine learning, image processing etc [18, 21, 27, 30]. In commerce, clustering can help to identify distinct group of users depending on their purchases. Depending on these clusters, more effective offers can be served. In biology, it can be useful to find genes with similar characteristics on particular symptoms. Clustering can be effective to identify similar patterns on land to analyze dispersion of life zones.

Clustering is an old research topic but it is still under development. There are basic methods for this purpose, but they are enhanced as data gets bigger and more complex. These methods can be explained under various titles. I explain them depending on data type they address. One type of methods clusters same type of objects, for example set of integers, strings or coordinates etc. Such data is called homogeneous. Each node in data is evaluated in the same manner. If data is composed of distinct types, for example set of age, nationality, profession, then it is heterogeneous.

2.3.1 Homogeneous Clustering

Each item in input can be treated as same type and grouped due to their similarity. This is called homogeneous clustering.

One of the most popular and commonly used approaches is distance based. The data is partitioned as close items are placed into same partitions. Natural language processing is the area where distance based clustering methods are widely used. It enables to group documents due to its concept or common ideas or topics addressed. It is highly necessary for librarians when scale of documents are considered. To fulfill this need, Dhillon and Modha have applied k-means algorithm on document vectors to make concept decomposition [10]. k-means algorithm is one of the popular distance based partitioning algorithm. By this method, thousands of documents can be evaluated and grouped simultaneously depending on their conceptual ideas.

Bioinformatics is another area where clustering methodology is highly used. Genome data is quite big and complex. It is suitable for many type of clustering. One of the works is k-means clustering on gene expressions [34]. Because it has many parameters and gene expression data is quite big, clustering methods may give very different results. But the results are promising for future researches.

2.3.2 Heterogeneous Clustering

Homogeneous clustering does not fulfill needs for many real world systems, because data gathered from them are complex. They are generally heterogeneous. It means data is composed of different type of items. Homogeneous clustering may discard relations among nodes from distinct types. This results lack of information at the end. On the other hand, on last decades, there has been an increasing interest to data mining as computing power increases. There are powerful resources, they can process many operations on data blocks simultaneously. Together with this, data becomes huge, especially with the help of social platforms. Therefore, many researchers have interests to extract more deep and relational information from them. Since they are directly trace of humans, many

organizations such as governments, marketers etc. have interests to make more discovery on this area, too. Considering these reasons, more complex clustering approach are necessary. Heterogeneous clustering enters the concept at this point. It helps to evaluate different types. By considering relations between nodes, it clusters of different type of nodes.

Under this discipline, multi-partite clustering, also called as k-partite clustering where k refers to number of partite groups inside a cluster has a significant place. It is applied on a graph data such that edges are not established among nodes from same types but from distinct types. In an obtained cluster, there are different number of partite groups, where each one represent a type. As a corollary, edges are only present between these groups. None of nodes inside same groups are adjacent.

Multi-partite clustering methodology has some advantages compared to homogeneous one. Because there are different types in clusters, nodes characteristics are more precisely determined. Rather than guessing similarity depending on being close to a node with same type, relations of nodes with same and different types are interpreted. This gives stronger meaning about that node. Therefore, multi-clustering is an early research topic and applied many disciplines. As an example for marketing, as people become more active in digital world, marketers aim to do customer analysis to understand needs of customers better. At the beginning, products similar to what bought are offered. As bi-clustering method is followed, first they find customers who buy same product. Then, depending on other purchases, people are offered to buy them. This is how recommendation engines entered and evolved in marketing [20].

Main disadvantage of multi-partite clustering is that it requires high computing power. Time complexity is directly proportional to the number k for k-partite clustering. As k increases, the complexity increases as well. Additionally, the main approach to clustering is that the size of clusters should be as big as possible, together with that the commonness inside clusters are preserved. However, mining maximum multi-partite clusters is proven to be NP-hard [8]. Therefore, researchers mostly aim to find considerably big clusters.

2.3.2.1 Bi-partite Clusters

As an entrance step to heterogeneous clustering, bi-partite clustering is a method to group nodes from two distinct types. Relations only exist between nodes from different types. The data used as input to this method can be also represented as bi-partite graph (Figure 2.6). This type of graphs are composed of two different type of elements, such as <user,photo>. If a user likes a photo, then an edge is established. But a user cannot like a user, also same for photos. If types are increased, then there will be more groups inside a graph. In the same manner, edges will be established only among nodes from different groups. This will be explained in subsection 2.3.2.2.

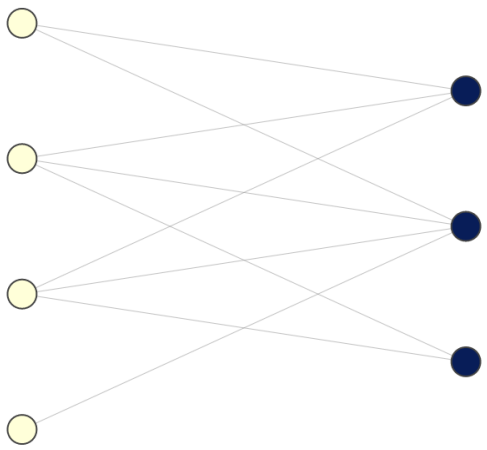


Figure 2.6: A Bi-Partite Graph

Bi-partite clustering is widely used in different fields. Natural language processing is an area where significance of biclustering has been demonstrated [9] by Dhillion. He has applied biclustering method to cluster documents and words simultaneously. He showed that his algorithm is generic and can be applied to any language. Depending on this base, Wieling and Nerbonne follow biclustering approach to do sensible geographical grouping of people depending on dialect pronunciation data for Dutch language [32].

2.3.2.2 Multi-partite Clusters

Multi-partite clusters are basically composed of groups where edges are established among groups, not inside groups. Nodes inside same groups in a cluster are highly similar. If the number of groups is 2, then this will be a bi-partite cluster; if 3, then tripartite and it goes on. This methodology is very useful and preferable for many real world scenarios. Because the data gathered from these systems are generally heterogeneous and complex, homogeneous clustering methodology [7, 23] is not sufficient to cover relations among different types. As contrast, multi-partite clusters keeps relations among heterogeneous data. It also keeps similar node inside same groups. Considering these features, this type of clustering methodology gives information from both inside of each group separately and among them. To illustrate this point, let us consider Instagram. There are many actions in it. One of the actions is that a user drops a comment under a photo. In this action, there are three nodes <user,comment,photo> and an hyperedge which connects all, which is the action itself. A set of this type of actions will be set of three different groups <user, comment,photo>. These groups are only interconnected. When this action set is represented as a graph, this graph will be multi-partite or k-partite graph where k is 3 (Figure 2.7). Tripartite clusters found on this graph will be helpful to analyze similar actions. Such composite information can be interpreted in many ways. It can be interpreted presumably as each node inside same group has a similar point of view. Another way of analyzing is considering groups due to other groups as parts of same cluster. For a user group, this method may give following ideas: They like similar photos. They use similar words for commenting on a photo from a particular genre. The photos which are commented by a group of users are instances of artistic photography. Then, these users are interested in photography art. The number of examples can be increased. The main point is that multi-partite clustering method provides deeper information as the reflection of real world actions.

Although multi-partite partite clustering is an old terminology, it becomes more popular in last decades together with social platforms. Social platforms produces

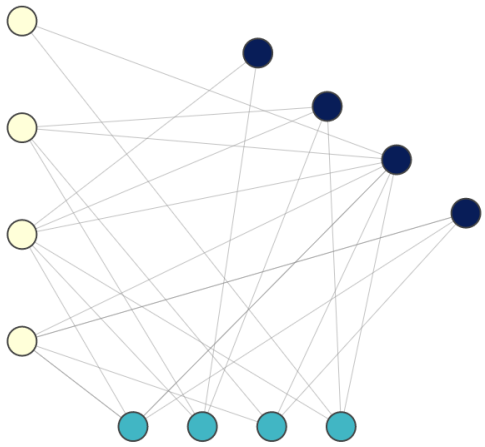


Figure 2.7: A k-Partite Graph ($k = 3$)

many composite relations. Experts from different areas want to analyze it to enhance their services. For example, research of Grinberg et al. analyze user actions at Facebook[1] platform to propose better feed to them [14]. The purpose is to guess what people would like to see on their feed. Hiding the unnecessary posts are crucial. However, the concept of being unnecessary is so subjective. It can significantly change depending on cultures, nations, age etc. At this stage, multi-partite clustering is helpful to identify similar users by grouping them depending on their actions, likes, who they follow etc. Then, similar strategy can be applied to arrange feeds of these users.

In this thesis, I will be addressing tripartite hypergraphs to find clusters on. This input graph is composed of relations where each connects three nodes from three different types. Therefore, each relation is a hyperedge with cardinality 3. There is an important difference between tripartite graphs and hypergraphs which our problem addresses. In tripartite graph, an edge can be established among two nodes independently from other third type. However, in the input hypergraph to the problem, if there is a connection between two nodes, then this connection must include a node from other third type, too.

Main drawback of multi-partite clustering is that mining multi-partite clusters

[1] https://www.facebook.com/

from such highly connected data brings performance issues. Furthermore, as type of nodes increases, finding clusters with maximal size will be more costly. Considering these points, for two types of nodes, searching for a bipartite graph with maximum size is proven to be NP-hard. In the same manner, mining three-partite clusters will be harder as increasing number of partitions will dramatically increase the hardness respectively [8]. Therefore, works in literature generally apply heuristics to find such clusters [25, 26, 35, 36]. Our solution also uses heuristics to find maximal clusters.

Heuristic approaches: As a common strategy, tripartite clusters are generated by first constructing bi-clusters between each pair of three partite groups [36]. Then, each bi-cluster is matched with two others in order to construct tripartite clusters. This approach requires evaluating same nodes repeatedly, though. As another alternative, Zhao and Zaki [35] first select two dimensions, and, then, discover bi-clusters for each node in third dimension. Then, it looks for intersection among bi-clusters of different nodes from 3^{rd} dimension. If an intersection is found, this part is claimed as a tripartite cluster. However, since the first two partitions are fixed, this approach has bias against the third partition.

Dhillon has approached differently on his work about co-clustering documents and words [9]. It is one of the early adaptations of multi-partite clustering to this area. In this work, he applied spectral co-clustering method on data and finding communities between words and documents. For partitioning, the heuristic is weight of vertices. One option is that all vertices have equal weight such as 1. This is called ratio cut [15]. Another option is that weight of a vertex is equal to sum of weights of edges incident on it. This is called normalized cut [31]. In this approach, it is aimed to find clusters depending on given input graph characteristics. Optimal partitioning is done if outgoing edges are minimum among all edges for each vertex set of two partitions. Cut approach has some drawbacks considering the problem we address. In order to find tripartite clusters for our problem, they must satisfy some density ratio values for negative and positive hyperedges. Cutting a graph could dramatically change the ratio values. However, for clusters whose values just pass the threshold ones, cutting operation should be performed delicately. Moreover, in spectral clustering nodes

can belong to only one cluster. It avoids node overlap.

Another approach is applying partitioning algorithm such as k-means. It is a popular method under clustering discipline. On social tagging system research, Lu et al. aim to find tripartite clusters inside a graph composed of three type of nodes; users, tags, resources [26]. He has adapted k-means approach to find such clusters inside a social tagging system. Their approach depends on connectivity of heterogeneous nodes. Furthermore, they do not consider weighted edges. In this manner, our problem is different.

As Internet has become a part of daily life, products or marketers looks for ways to address people more effectively. This is where recommendation systems become important. Such systems are essential to offer items due to users' interests. It is also a popular research area. Work of Cheng et al. is a good example of adaptation of k-partite graphs in order to provide such specialty [5]. In this work, each pair of node type has an adjacency matrix. Then, by non-negative factorization approach, clustering is done. This approach does not carry out density ratio properties of given graph. Second example is the tagging system. There are platforms where users can attach tags on items. But there can be some ambiguities on these tags, such as an abbreviation used as tag can refer to distinct meanings. Yeung et al. apply tripartite structure to discard ambiguities [33]. GN approach, introduced by Girvan and Newman [29], is applied on graph data to find tripartite communities. This modularity-based approach focuses on communities in which nodes are strongly connected, but between communities are loosely connected. However, clusters found are various. They can be homogeneous cluster as well as heterogeneous ones. It does not consider density ratios, too. When it is considered that our aim is to find tripartite clusters with certain specialties, this approach is not very suitable.

Another example is that Zhao and Zaki introduce TriCluster algorithm to cluster three different types: genes, samples and time slots [35]. To start clustering in heterogeneous data, source of the motivations is the effectiveness and adaptability of bi-clustering method [6]. Second, the results which comes from applying bi-clustering on microarray data are very promising [12]. Depending

on this approach, first ratios for each pair of gene and sample are calculated. From valid ratio-ranges, range multigraph is constructed. They search maximal bi-clusters of genes and samples depending on these multigraphs. Finally, maximal tripartite clusters are constructed based on the bi-clusters. Hyperedges in same tripartite clusters have relative ratios. They are the rates which stand for the characteristic of genes. If a gene cannot follow same ratio pattern, then it is out of that cluster.

TriCluster algorithm looks for clusters with different characteristics. Hyperedges in these clusters follow same pattern of existential ratio for time dimension. Positive or negative property is different.

Overlapping: One of the main considerations in multi-partite clustering is whether a node can be a part of many clusters or not. In the input graph, a node can be connected via many hyperedges. So, both approach is possible. However, if it is contained by only one (one-to-one correspondence), it brings less number of clusters with small sizes [22, 24]. Because real world data is generally more complex and bigger, a node is more likely to be a part of many hyperedges and also clusters. It requires a node to be connected with many nodes from other types (many-to-many correspondence). For the example of social-tagging system, a user can tag many resources with using many keywords. If a user can be a member of only one cluster, then for this user, interactions out of the cluster are discarded. This results and incomplete analysis of this user. Therefore, it will be more informative if clusters share nodes. This is named as node overlapping.

Including node overlapping, hyperedges can be shared among clusters, too. This may be an option where hyperedges are so much important and they put a great value on clusters. Such example is TriCluster algorithm [35]. In this algorithm, clusters are constructed depending on existence ratio of a gene in samples at particular times. Therefore, a set of hyperedges of <gene,sample,time> triple can be contained by many clusters. For such approach, hyperedge overlapping could be crucial not to miss any important cluster. In our problem, node overlapping is desired, but hyperedge overlapping needs to be avoided.

CHAPTER 3

MINING 3-PARTITE CLUSTERS

Our solution is based on 3 different constraints. These constraints depends on five user-defined parameters. Three of these parameters are for controlling the sizes of each dimension of the clusters and the other two parameters are used to control the positive/negative hyperedge ratios in clusters. These input parameters, together with related cluster features and the definitions of the size and the ratio constraints, are given in Table 3.1. Since, the positive sign of the hyperedge means three nodes forming this edge agree, and the negative sign means they disagree, the idea is to form clusters on three dimensions using this agreement relationship and maximizing the agreement while minimizing the disagreement among the nodes of co-clusters. Furthermore, we want to generate the largest possible clusters possible. So, we have two conflicting objectives. For a given tripartite graph, we would like to construct the largest possible tripartite clusters, as well as generate clusters with maximum density of positive signed hyperedges and minimum density of negative signed hyperedges. There is a trade-off between these two objectives. By trying various values of the parameters in Table 3.1, clusters with different properties/qualities may be generated. As expected, only very small size or trivial perfect clusters can be generated with full positive labeled edges. In order to generate more meaningful and useful clusters with larger sizes, we may tolerate some negative labeled edges in clusters and reduce the minimum positive edge ratio requirement.

To simplify the process, we have generated a fully connected tripartite graph by adding hyperedges with no sign between all 3 nodes from 3 different dimensions,

Table3.1: Cluster Control Parameters and Constraints

Symbol	Meaning
Input Parameters	
ϵ_p	minimum h_+ edges ratio parameter in a cluster
ϵ_n	maximum h_- edges ratio parameter in a cluster
λ_i	minimum size of dimension i parameter in a cluster
Cluster Properties	
L_i	number of nodes for type i in a cluster (size of type i)
C_p	number of h_+ edges in a cluster
C_n	number of h_- edges in a cluster
Constraints	
$\epsilon_p \leq \frac{C_p}{L_1 \times L_2 \times L_3}$	ratio of h_+ edges constraint in a cluster
$\epsilon_n \geq \frac{C_n}{L_1 \times L_2 \times L_3}$	ratio of h_- edges constraint in a cluster
$L_i \geq \lambda_i$	size for dimension i constraint in a cluster

if they are not already connected in the original graph with negative or positive signed edges. We have developed a simple and efficient greedy heuristic in order to generate clusters satisfying above mentioned constraints. The general idea is to start with the whole graph as a cluster, then trim less effective nodes until ratio constraints are satisfied, or it cannot satisfy the size constraint. Effectiveness of the nodes is also defined using simple formulas which will be discussed below. Our method works as follows:

1. Start with a single co-cluster of the whole graph and trim the least effective node from it in order to increase its positive density and decrease its negative density as much as possible. Notice that this trimming operation reduces the size of the cluster.

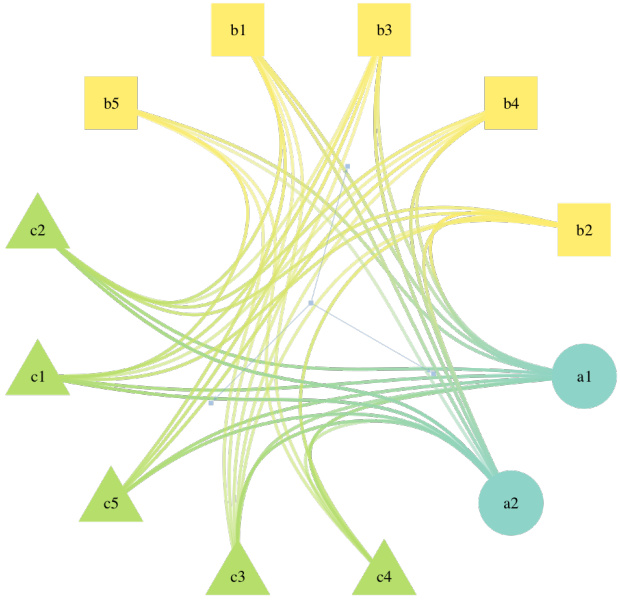
2. Repeat this trimming operation until a cluster is obtained satisfying both the density constraints and the cluster size constraints, or until one of the minimum cluster size constraints are violated.

    - If the obtained cluster satisfies all the constraints, it is added to the cluster list. Then, in order to remove the hyperedges used in this cluster, least effective nodes incident to each hyperedge is removed

a_1	b_1	b_2	b_3	b_4	b_5
c_1		−		+	−
c_2	−	+	+	+	
c_3	+				+
c_4		−			+
c_5	+		−	+	

a_2	b_1	b_2	b_3	b_4	b_5
c_1		+	−		+
c_2	+	+		+	
c_3	+	+	−		−
c_4	−	+	+		+
c_5	+				

(a) Matrix Representation



(b) Graph Representation

Figure 3.1: Input Data

from the graph. The process then repeats itself from the beginning using the remaining graph.

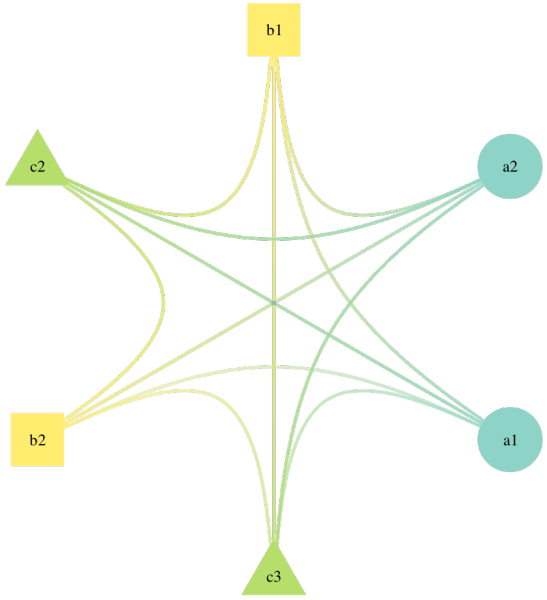
- If the obtained cluster violates one of the size constraints, all node removals from this iteration were backtracked, and only one hyperedge with negative sign is selected from the graph and one of the nodes incident to that edge is removed from the graph. The process repeats itself from the beginning using the remaining graph.

In this paper, we use the notations given in Table 3.1. STRICLUSTER algorithm takes a set of hyperedges, Γ as an input, such that each hyperedge h connects three nodes from three different types $U = (U_1, U_2, U_3)$. Figure 3.1

a_1	b_1	b_2
c_2	−	+
c_3	+	

a_2	b_1	b_2
c_2	+	+
c_3	+	+

(a) Matrix Representation



(b) Graph Representation

Figure 3.2: Output Cluster, when ϵ_p=0.75, ϵ_n=0.15

illustrates hyperedges given as 3D matrix. These hyperedges have either positive or negative labels, represented by positive and negative signs respectively in Figure 3.1. Remaining entries (white cells) correspond to node triples without connecting hyperedges. In the example, nodes are $\{\{a_1, a_2\}, \{b_1, b_2, b_3, b_4, b_5\}, \{c_1, c_2, c_3, c_4, c_5\}\}$ from types U_1, U_2, U_3 respectively.

The aim of STRICLUSTER is to find tripartite clusters of hyperedges with highly positive labels. To be a valid tripartite cluster, it has to satisfy threshold values for both density and size. The density threshold values are ϵ_p and ϵ_n, such that $0 \leq \epsilon_p, \epsilon_n \leq 1$, $(\epsilon_p + \epsilon_n) \leq 1$. The former one represents the minimum ratio density of positive hyperedges (h_+) among all possible hyperedges (i.e., there may be $L_1 \times L_2 \times L_3$ number of possible hyperedges for a cluster with size (L_1, L_2, L_3), where L_i is number of nodes with U_i type in the cluster). If C_p is

the number of h_+, then:

$$\epsilon_p \leq \frac{C_p}{L_1 \times L_2 \times L_3}, \tag{3.1}$$

If $\epsilon_p = 1$, generated tripartite clusters become tripartite cliques as well. ϵ_n is the value to control the density of negatively signed hyperedges (h_-). If C_n represents the number of h_-, then:

$$\epsilon_n \geq \frac{C_n}{L_1 \times L_2 \times L_3}, \tag{3.2}$$

shows maximum allowed tolerance of h_- in a cluster if $\epsilon_n \neq 0$.

In order to prevent constructing very small clusters λ_i is defined, such that:

$$L_i \geq \lambda_i, \tag{3.3}$$

for $1 \leq i \leq 3$, $i \in \mathbb{N}$. i stands for the corresponding dimension. This constraint should also be satisfied by every cluster.

An example cluster obtained from the input graph given in Figure 3.1 is depicted in Figure 3.2.

STRICLUSTER algorithm (Algorithm 1) starts by generating a potential cluster α which contains all hyperedges in Γ. The main loop (lines from 4 to 24 in Algorithm 1) iteratively constructs clusters satisfying both size and density constraints.

It begins with the remaining nodes of the graph and removes least effective nodes (through while loop in lines 6 to 9), until either all density constraints are satisfied (conditions at lines 6 and 7), or until the graph becomes too small to satisfy minimum size constraints due to these node removals.

If both the density and size constraints are satisfied, the remaining nodes of the graph form a cluster which is added to the cluster list (at line 12). In addition, its edges (all signed edges) are removed from the graph (by adding them to invalid hyperedge list at lines 13 and 15), so they cannot be used in the construction of a new cluster. This way guarantees clusters will not share edges, but can still share nodes.

As mentioned above, due to trimming of the graph by removing its least effective nodes to satisfy density constraints, the graph may become too small and unable

Algorithm 1 STriCluster Algorithm

1: **procedure** STRICLUSTER($\Gamma, \epsilon_p, \epsilon_n, \lambda_1, \lambda_2, \lambda_3$)
2: $\Gamma' \leftarrow \Gamma$
3: $\Re \leftarrow \emptyset$
4: **loop**
5: $\alpha \leftarrow \Gamma'$
6: **while** (SIZECHECK($\alpha, \lambda_1, \lambda_2, \lambda_3$)) **and**
7: **not** DENSITYCHECK($\alpha, \epsilon_p, \epsilon_n$) **do**
8: REMLEASTEFFNODE(α)
9: **end while**
10: **if** DENSITYCHECK($\alpha, \epsilon_p, \epsilon_n$) **and**
11: SIZECHECK($\alpha, \lambda_1, \lambda_2, \lambda_3$) **then**
12: $\Re \leftarrow \Re \bigcup \{\alpha\}$
13: **for each** $h_{-/+} \in \alpha$ **do**
14: $\Gamma_{iv} \leftarrow \Gamma_{iv} \bigcup \{h_{-/+}\}$
15: **end for**
16: **else**
17: $h_- =$ RANDOMNEGATIVEEDGE(α)
18: $\Gamma_{iv} \leftarrow h_-$
19: **end if**
20: CLEANINVALIDS($\Gamma, \Gamma_{iv}, \Gamma'$)
21: **if not** SIZECHECK($\Gamma', \lambda_1, \lambda_2, \lambda_3$) **then**
22: **return** $\Re$
23: **end if**
24: **end loop**
25: **end procedure**

to satisfy the size constraints any more, and thus, the else branch of the if statement between the lines 16 and 19 of the algorithm is executed. A very simple heuristic is done here. A negative edge is looked for in α, and the one first found is selected and added into the invalid edge set. This way, the next round of iteration of the main loop starts with a graph with one less negative edge, and the chance for generating higher positive density clusters increases.

After this main if statement (from lines 10 to 19 in the algorithm) either one cluster is generated and its edges are added to the invalid edge list, or just one negative edge is added to the invalid edge list. Then, these invalid edges must be removed from the graph. This is done by removing one of the nodes incident to these edges in order to be able to reduce the graph size as well. The procedure at line 20 removes one node for each invalid edges from the original graph, then the remaining graph (Γ') is used in the next iteration to discover another cluster from it. However, the remaining graph may be too small, and if it does not satisfy the size constraints (checked at line 21), the process ends and clusters obtained so far are returned (at line 22).

3.1 Density Check

DensityCheck is a procedure with 3 input parameters. They are a potential cluster α and two constraints values ϵ_p and ϵ_n. The purpose of this sub-procedure is controlling quality of given cluster due to constraints 1 and 2. If both constraints are satisfied, then this sub-procedure returns **true**. Otherwise, it returns **false** to point out that removing a node is necessary.

3.2 Size Check

SizeCheck is a procedure with 4 input parameters. The first one is a potential cluster α, the others are size constraints for each of 3 dimensions. This sub-procedure checks the size of given cluster. If size of each dimension is not below of its corresponding limit (constraint 3.3), then this sub-procedure returns **true**.

Otherwise, it returns **false**.

3.3 Remove Least Effective Node

RemLeastEffNode procedure takes one input parameter which is a potential cluster α. It removes a node from α.

a_1	b_1	b_2	b_3	b_4	b_5
c_1		$-$		$+$	$-$
c_2	$-$	$+$	$+$	$+$	
c_3	$+$				$+$
c_4		$-$			$+$
c_5	$+$		$-$	$+$	

a_2	b_1	b_2	b_3	b_4	b_5
c_1		$+$	$-$		$+$
c_2	$+$	$+$		$+$	
c_3	$+$	$+$	$-$		$-$
c_4	$-$	$+$	$+$		$+$
c_5	$+$				

Figure 3.3: Removing Node b_3

Heuristic calculations are used to determine which node to remove. There is a simple approach behind this. If a node is connected by high number of h_+, it should be less likely to be removed. If a node is highly linked by negatively signed hyperedges or it is loosely connected, its probability of being removed is high. Due to the statement of the problem, positively labeled hyperedges are valuable. Therefore, it is not desired to extract them from a cluster. In order to keep them inside, they are valued by a positive number. In contrast, a negative number is given the ones carrying negative labels (Equation 3.4).

$$val(h) = \begin{cases} 1 & \text{if } h \text{ has positive label} \\ -1 & \text{if } h \text{ has negative label.} \end{cases} \tag{3.4}$$

With the help of it, effectiveness of each node is determined. For this purpose, Equation 3.5 is used.

$$E_{ir} = \frac{1}{S_i} \times \sum_{h \in \alpha} \begin{cases} val(h) & \text{if } r \in h \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

E_{ir} is the effectiveness value of node r from type i. It is a density calculation of hyperedges which connect a node to the cluster. S_i refers to maximum number

of hyperedges which can be linked to that node. For each type, S_i value differs (Table 3.2). The lowest effectiveness value gives the node to remove. An example is given in Figure 3.3, where the least effective node is b_3.

Table3.2: Maximum Possible Number of Hyperedges for Each Type

Type	Value
S_1	$L_2 \times L_3$
S_2	$L_1 \times L_3$
S_3	$L_1 \times L_2$

3.4 Random Negative Edge

RandomNegativeEdge is a procedure taking 1 input parameter, which is a potential cluster α. This sub-procedure helps to invalidate a hyperedge. It finds and returns one which is not significant whether it is covered or not. Predictably, it is the negatively signed one. The sub-procedure traverses all hyperedges and returns the first h_- it finds.

3.5 Clean Invalids

Invalid hyperedges are not desired to be part of future clusters. The hyperedges of all tripartite clusters previously found are invalid. Moreover, hyperedges returned by RandomNegativeEdge procedure are added to the list of invalids. CleanInvalids (Algorithm 2) procedure aims to remove all invalid hyperedges inside potential cluster Γ'. Γ is copied as Γ' at the beginning of this procedure. As a result, Γ' does not contain any hyperedge in Γ_{iv} (the list of invalids). This procedure returns. Then, the algorithm searches a valid cluster inside Γ'.

In order to remove a hyperedge, one of the nodes linked by this hyperedge should be removed. In this manner, to remove an invalid hyperedge, CleanInvalids procedure looks for a node, then removes. It repeats the same removing action until no invalid hyperedge is left in Γ'. While selecting a node, it uses a heuristic

Algorithm 2 Clean Invalids Procedure

1: **procedure** CLEANINVALIDS($\Gamma, \Gamma_{iv}, \Gamma'$)
2: $\Gamma' \leftarrow \Gamma$
3: CALCULATE(γ)
4: **while** $max(\gamma) \neq 0$ **do**
5: $\gamma_x = max(\gamma)$
6: $\gamma \leftarrow \gamma \ominus \gamma_x$
7: $\Gamma' \leftarrow \Gamma' \ominus h_x$
8: UPDATE(γ) ▷ affected from x
9: **end while**
10: **end procedure**

that reduces side effects of removing nodes on the cluster size as much as possible.

On the heuristic calculation, first, the number of invalid hyperedges connected to each node is counted. This value is then divided by S_i where i refers to the type of that node (Equation 3.6). The final value θ_{ir} is the density of invalids for that node. If θ_{ir} is high, the node r is more likely to be removed.

$$Q_{ir} = \frac{1}{S_i} \times \sum_{h \in \alpha} \begin{cases} 1 & \text{if } r \in h \wedge h \text{ is invalid} \\ 0 & \text{otherwise} \end{cases} . \tag{3.6}$$

Next, the effectiveness of each node in Γ' for its all valid hyperedges is calculated. This calculation is a modified version of Equation (3.5). Additionally it checks if hyperedges are valid. If not, they are not counted in the calculation (Equation 3.7). As another modification to Equation 3.5, add-1-smoothing approach is applied. Because it will be denominator on the next calculation (Equation 3.8), it is guaranteed that this value will never be negative. The final value E^v_{ir} is the effectiveness value for node r. A node with high E^v_{ir} will likely not be excluded.

$$E^v_{ir} = \frac{1}{S_i} \times \sum_{h \in \alpha} \begin{cases} 1 + val(h) & \text{if } r \in h \wedge h \text{ is valid} \\ 0 & \text{otherwise} \end{cases} . \tag{3.7}$$

θ_{ir} is directly proportional with selecting a node to remove, as E^v_{ir} value is inversely proportional. Therefore, the final calculation is performed as in Equation

3.8. c is a constant value which is generally set to 1.

$$\gamma_{ir} = \theta_{ir} \times \frac{1}{c + E^v_{ir}}. \tag{3.8}$$

Among all nodes, the one with the highest γ value is removed (from line 5 to 7). Then, γ values for remaining nodes inside Γ' are updated, excluding the ones from same type with recently removed node x. This update operation is performed for the nodes whose γ value calculations count hyperedges omitted via removing node x. It is done as follows. Q_{ir} and E^v_{ir} values for each node r from type i is stored. For each node, these values are reduced by subtracting recently omitted hyperedges. Then, Equation 3.8 for each recalculated again. Then, this procedure continues removing node with maximum γ value.

If a node is not linked to any invalid hyperedges, then Equation 3.6 will result 0 for that node. This causes γ value for this node to be 0, too. By this manner, if maximum γ value is 0, then it means there is no invalid hyperedges in Γ' (line 4). This procedure returns. Removing a node continues until this state will be reached (from line 4 to 9).

For the input data in Figure 3.1, STRICLUSTER algorithm finds the cluster in Figure 3.2 in the first iteration. Then, hyperedges of this newly generated cluster are invalidated. To construct a new potential cluster, CLEANINVALIDS procedure is called. At the beginning, Γ' (Figure 3.4a) is generated from Γ. But Γ' contains some invalid hyperedges (colored with purple in Figure 3.4a). To find a node to remove, γ values are calculated. First, node b_1 is removed since γ_{3b_1} is $(4 \div 10) \times (10 \div 14) \cong 0.29$ which is the maximum among γ values. Then, node b_2 is selected and removed (Figure 3.4b). This will result a clean Γ' (Figure 3.4c) and the procedure terminates.

3.6 Complexity Analysis

Worst case scenario for STRICLUSTER algorithm is that no tripartite cluster is found. In this case, in each iteration RANDOMNEGATIVEEDGE is called to throw an h_e. Then, this hyperedge is invalidated. It means that hyperedges in Γ

a_1	b_1	b_2	b_3	b_4	b_5
c_1		$-$		$+$	$-$
c_2	$-$	$+$	$+$	$+$	
c_3	$+$				$+$
c_4		$-$			$+$
c_5	$+$		$-$	$+$	

a_2	b_1	b_2	b_3	b_4	b_5
c_1		$+$	$-$		$+$
c_2	$+$	$+$		$+$	
c_3	$+$	$+$	$-$		$-$
c_4	$-$	$+$	$+$		$+$
c_5	$+$				

(a) Removing First Node

a_1	b_2	b_3	b_4	b_5
c_1	$-$		$+$	$-$
c_2	$+$	$+$	$+$	
c_3				$+$
c_4	$-$			$+$
c_5		$-$	$+$	

a_2	b_2	b_3	b_4	b_5
c_1	$+$	$-$		$+$
c_2	$+$		$+$	
c_3	$+$	$-$		$-$
c_4	$+$	$+$		$+$
c_5				

(b) Removing Second Node

a_1	b_3	b_4	b_5
c_1		$+$	$-$
c_2	$+$	$+$	
c_3			$+$
c_4			$+$
c_5	$-$	$+$	

a_2	b_3	b_4	b_5
c_1	$-$		$+$
c_2		$+$	
c_3	$-$		$-$
c_4	$+$		$+$
c_5			

(c) Clean Γ'

Figure 3.4: Cleaning Invalids

will be invalidated one by one in each iteration. Therefore, the outer loop (line from 4 to 24) in STRICLUSTER (Algorithm 1) will iterate $(L_1 \times L_2 \times L_3)$ (n refers to this value) times. In each iteration, the loop between lines 6 and 9 removes all nodes one by one until SIZECHECK returns **false**. It is $(L_1 + L_2 + L_3)$ number of iterations. Additionally, for effectiveness value calculations of nodes, all hyperedges with a sign are visited. Total hyperedge count depends on the input data. However, it can be n at maximum. Finally, time complexity of loop in lines 6 and 9 is $\mathcal{O}(n) = n \times (L_1 + L_2 + L_3)$. Similarly, CLEANINVALIDS procedure does node removal depending on hyperedge values. Therefore, its complexity is also $\mathcal{O}(n) = n \times (L_1 + L_2 + L_3)$. As a result, time complexity of STRICLUSTER algorithm in worst case is $\mathcal{O}(n^2 \times (L_1 + L_2 + L_3))$.

In best case, whole input is one cluster. Then, the most outer loop only iterates once. Additionally, node removing operation is not performed at line 8. But all nodes are removed in CLEANINVALIDS, since all hyperedges are contained by previously found cluster. Then, the algorithm concludes at line 22, because SIZECHECK is not satisfied for Γ'. This shows that time complexity of the algorithm in best case is $\mathcal{O}(n \times (L_1 + L_2 + L_3))$

Considering space usage, STRICLUSTER algorithm holds whole input in memory in a 1D array. Size of this array is $n = (L_1 \times L_2 \times L_3)$. Γ' holds hyperedges whose count has upper bound n. Similarly, Γ_{iv} can have a hyperedge only once. Therefore, its size is limited to n, too. $\Re$ is the list of obtained clusters. Since there is no hyperedge shared by clusters, each hyperedge can only be present once inside $\Re$. Therefore, it contains at most n number of hyperedges. Additionally, heuristic values for each node are hold in memory. There can be $2 \times (L_1 \times L_2 \times L_3)$ values at maximum. This analysis shows that STRICLUSTER algorithm works in linear space. Its space complexity is $\mathcal{O}(n)$.

CHAPTER 4

EXPERIMENTS

Two different types of experiments have been performed to test STRICLUSTER algorithm. The first one is done on a synthetic data. In the second one, a real-world data is used.

4.1 Synthetic Data

In this case, input data is generated artificially. The densities of positive and negative hyperedges in it are under control. Depending on data size and custom parameters which are passed to STRICLUSTER algorithm, different test cases are defined. This test is convenient to see how the algorithm performs on data with different densities. Moreover, for different number of positive and negative hyperedges, behavior of the algorithm is observed.

4.1.1 Data Acquisition

Generated input data is a 3-D matrix with three cell types: +, - or empty. Additionally, some rules are followed. Firstly, h_+ and h_- in input data have density ratios. After input generation, it is ensured that the data contains h_+ and h_- with corresponding density ratios. As another rule, we fix these ratios while changing input size. Other parameters are constant in these scenarios, which are listed on Table 4.1.

In order to briefly mention the constants in Table 4.1, the constraints in these

Table4.1: Constants for Test Cases

Symbol	Value
ϵ_p	0.75
ϵ_n	0.10
λ_i	(2,2,2)

test scenarios can be explained respectively as follows. h_+ ratio of a cluster will be equal to or grater than 0.75. Ratio of h_- will be equal to or lower than 0.10. Dimension sizes of each cluster will be at least 2.

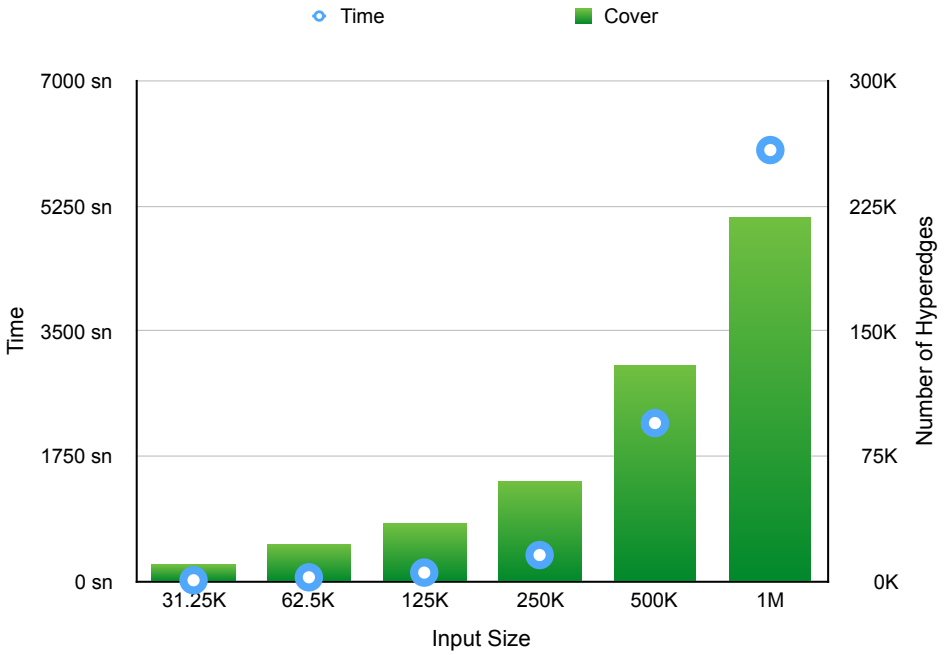
On test scenarios, STRICLUSTER algorithm is run on each sample data separately. After an execution is ended, two values are recorded. The first one is execution time of the algorithm (blue dots on Figure 4.1). The second one is the cover of nodes. The cover represents number of distinct hyperedges contained by clusters (green sticks on Figure 4.1). We have plotted these values for each cases.

4.1.2 Test on Inputs with Different Sizes

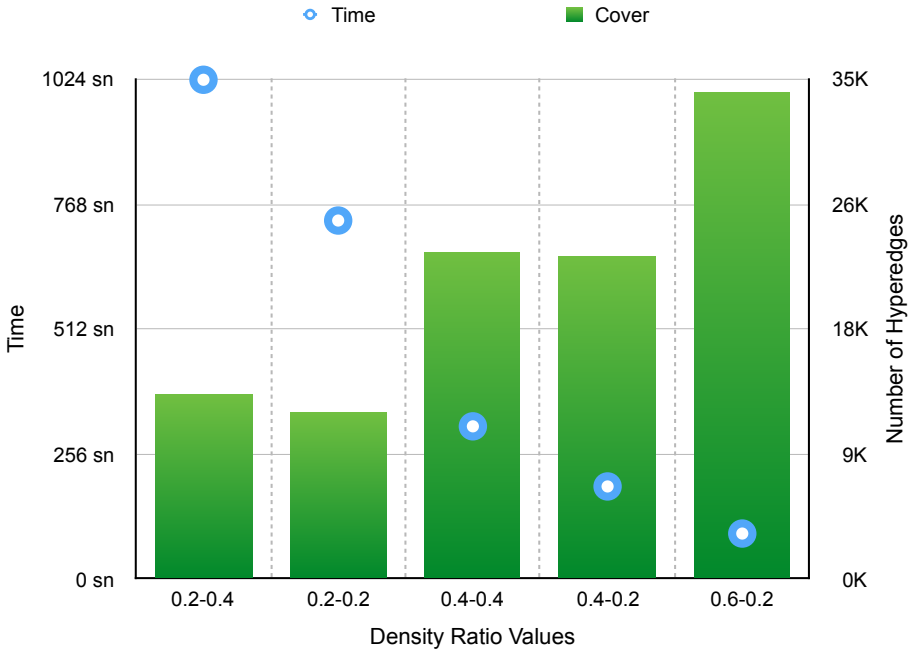
In the first test, we have 6 sample data. In 3D matrix format, each sample contains positive hyperedges with %60, negative hyperedges with %20. %20 of the input is empty. The distribution is done randomly. $(L_1 \times L_2 \times L_3)$ values, sizes of matrices, for these samples are (31.25K, 62.5K, 125K, 250K, 500K, 1M) respectively.

As shown in Figure 4.1a, consumed time values are exponentially increasing with n^2. This is expected due to heuristic calculations. Because all hyperedges of discovered clusters are invalidated, it is expected that number of hyperedges used for heuristic calculations will decrease fast. In the best case, this reduction will be logarithmic.

On the other hand, the number of hyperedges covered by clusters are directly proportional with size of input. In this test, the ratio of this number with size of input is about %20.



(a) Test on Inputs with Different Sizes



(b) Test on Inputs with Different Densities

Figure 4.1: Experiment Results on Synthetic Data

4.1.3 Test on Inputs with Different Densities

In the second test, we have 5 different sample data. We fix the input size as $(L_1 \times L_2 \times L_3) = 125K$. Density ratio values for h_+ and h_- are listed in Table 4.2.

Table4.2: Density Ratio Values for h_+ and h_- in Input Data

Sample No	Ratio for h_+	Ratio for h_-
1	0.2	0.2
2	0.2	0.4
3	0.4	0.2
4	0.4	0.4
5	0.6	0.2

Test results are plotted in Figure 4.1b. As density ratio of h_+ increases, execution time decreases, because bigger clusters can be obtained. Bigger clusters will fasten to consume valid hyperedges. It results that the algorithm ends quickly.

When density ratio of h_+ is low, the algorithm struggles more to discover a cluster. One of the reasons is that node removal is done one by one. In addition, obtained clusters will be smaller. This situation causes number of valid hyperedges to reduce slowly. On such case, by increasing size constraints (λ_i), mining small clusters can be avoided. This will also help the algorithm to conclude faster.

On the other hand, execution time is also directly proportional with density of h_- and sparseness in input data. Since h_- in clusters are also controlled by ϵ_n, more work is necessary when their number is high.

4.2 Real-World Data

Brexit[1] referendum is a very popular political event of 2016. The citizens of UK have voted if UK should leave EU or not. To effect opinions, many talks have been done by political parties. Heralds worked to emphasize people for their sides, not only on televisions and public talks, but also on social media. Social media is a very effective platform where vast number of people can be readily reached. Therefore, social media is particularly focused for sharing ideas about Brexit. Millions of tweets have been posted. Since there are great number of expressive tweets, it is easier to find numerous ones which have many common parts among themselves. This is why we have collected data about Brexit referendum from Twitter to run the algorithm on.

4.2.1 Data Acquisition

We collect tweets of 412 number of Twitter accounts which belong to politicians from 5 major political parties in United Kingdom. Then, we utilize Twitter Search API to get the latest 3200 tweets of each politician. For preprocessing, tweets dated before January 1, 2016 are removed.

To represent each politician's stance towards the issue in binary we utilize off-the-shelf sentiment analysis tool SentiStrength[2]. We assume that overall sentiment score of the tweet implies the opinion of the tweet towards the issue word that the tweet contains. Each sentiment is saved as either positive, negative or neutral. Additionally, an issue of a tweet and deterministic keywords in it are attached to the sentiment.

Major 5 political parties viewpoints can be summarized as below [17];

- Labour: Overwhelming majority of Labour Party members campaign for staying in European Union although there were raising concerns about the structure and function of the European Union.

[1] https://en.wikipedia.org/wiki/Brexit
[2] http://sentistrength.wlv.ac.uk/

- Conservatives: The leader of the Conservative Party, David Cameron offered the referendum and started the campaign for remaining in EU. There was a clear leaning towards leaving the EU despite the Cameron's efforts.
- Libdem: Liberal Democrats campaigned for staying in the EU.
- UKIP: UK Independence Party was a prominent figure in the referendum campaign. They passionately advocated to leave the EU. Blocking the refugees from entering the country, opposing international and EU-wide trade agreements, defending UK-born workers' rights over immigrants' rights were standing out as motivating factors in their campaign.
- SNP: Scottish National Party campaigns to stay in EU.

After sentiment analysis, we have $453,519$ tweets of 412 users. There are 36 different issues. The data contains $27,208$ distinct keywords. Occurrence of each keyword is counted. This data is so sparse. The fullness ratio is ~ 0.0011. In order to decrease sparseness, most frequent 1000 keywords are selected, by keeping number of users and issues stable.

Final size of input matrix is $412 \times 36 \times 1000 = 14,832,000$. The total number of hyperedges in it is $175,421$. The rest of it is sparse. Thus, the density of hyperedges in the input is ~ 0.011. $119,525$ number of them are with negative label. $55,896$ number of them are positive ones. Since the negative ones doubles the positives and the algorithm finds clusters with high positive density, the labels of hyperedges are switched.

4.2.2 Baseline Methodology

This work is done by Mert Özer from ASU.

As a baseline method we apply Tucker decomposition to Γ to find user, issue and sentiment word clusters in the Brexit data set. To that end, we utilize the Tucker decomposition component [19] of MATLAB Tensor Toolbox Version 2.6 of [1].

With the expectation to determine Remain and Leave camps, we have generated 2 clusters for the user dimension and 3 clusters for the issues and sentiment words dimensions, respectively. When two opposing camps exist, it is also likely that the other dimensions will have reflections of these two camps as two clusters and there may be a third cluster for the remaining issues and the sentiment words. Since spectral clustering includes all the nodes of all these dimensions, we have obtained very large clusters with very low edge densities. Notable results that we have observed from this experiment are as follows:

- Issues are unevenly distributed to 3 clusters. The first one contains three of the most popular issues, namely *citizenship, Brussels* and *worker*. There is a strong negative reaction towards these issues from the first user cluster. The next cluster contains 11 issues, and against those issues, there is less negative reaction from the second user cluster. This issue cluster contains issues like *humanrights, tuition, EU* etc. The reaction is not very clear on remaining clusters. Thus, it is not possible to obtain any useful information from issue dimension.

- Sentiment keywords have also been unevenly distributed across three clusters. Even the smallest cluster contains 105 sentiment words. Positive and negative sentiment words were also distributed through the clusters. There is no useful result that can be obtained from these clusters either.

- The user clusters obtained from this method are also not very informative. The first cluster contains 133 politicians from a variety of parties. The largest group in this cluster is Labours with 62 members, which is followed by 44 members of the Conservatives. It also contains 8 SNP, 8 Liberal and 5 UKIP members, as well as 6 members from other parties. The second cluster contains 239 politicians. Labours increase to 95, Conservatives almost double to 90, and other parties also increase, Liberals to 25, UKIP to 11, and SNP to 15. So, these clusters do not give any information about the party membership vs issue relationship either.

4.2.3 Test Results

Five different runs of STRICLUSTER have been performed with different values of input parameters λ_i, ϵ_p and ϵ_n. The values are listed on the Table 4.3 with respect to test numbers. λ_i values are for users, issues and keywords respectively.

Table4.3: Variables for Tests with Real Data

Case No	λ_i	ϵ_p	ϵ_n
1	(7,3,5)	0.7	0.07
2	(10,3,10)	0.5	0.05
3	(20,3,10)	0.5	0.05
4	(20,3,10)	0.3	0.03

Rather than performance of the executions, the focus is the analysis of output clusters on this test. The obtained clusters are interpreted by considering the sides and their opinions on Brexit. Due to the approach of the algorithm, people are not placed in same cluster by only considering their political views. Their opinions about specific issues make them members of same clusters. Furthermore, if users use identical keywords for specific issues, then they are more likely to be in same clusters. This is also helpful to discover which keywords are more popular to describe those particular issues.

Case 1: More dense clusters are aimed to be found. Minimum value for the density ratio of positive hyperedges are kept high. It is 0.7 in this test. The size thresholds are relatively low. Since the density threshold values are high, it is more likely to have small clusters. The sparseness of the input is also effective on this result. Small number of clusters are predicted. Consequently, only one cluster with size $17 \times 3 \times 6$ is found.

On Figure 4.2, 17 number of yellow nodes show users. They are all from labors group. Light green ones in square shape are issues, which represents *tax, NHS (National Health Service)* and *EU (European Union).* The blue ones in triangle shape are keywords. The keywords are *use, claims, hate, lost, things* and *chamber.* This clusters are very compact and dense. Therefore, it can be claimed

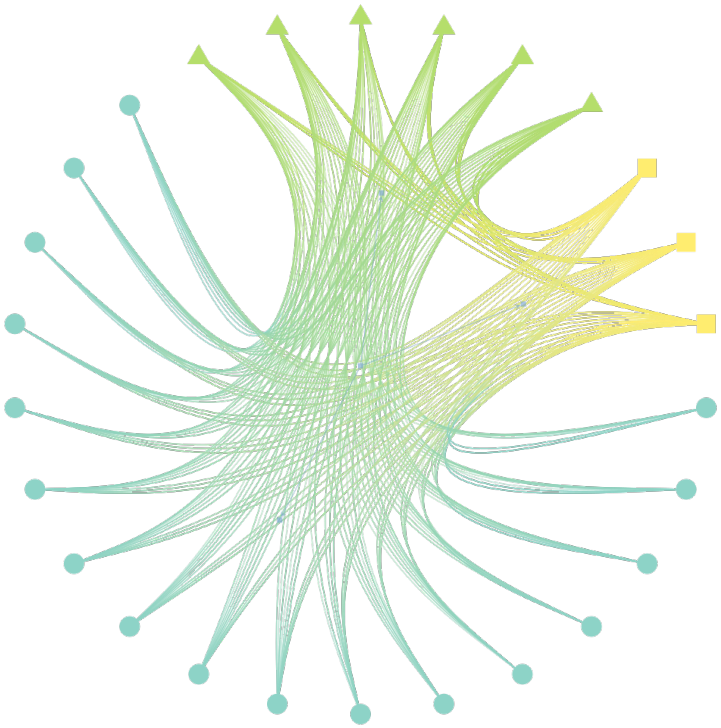


Figure 4.2: 1 Cluster in Case-1

that these users express quite similar opinions about the issues.

Case 2: In this case, density constraints ϵ_p and ϵ_n are lowered to 0.5 and 0.05 respectively. Relatively, size limits are increased(Table 4.3). Under these circumstances, the algorithm has found 3 different clusters (Figure 4.3). Four square nodes (colored as yellow) represent 4 distinct issues which are *tax, NHS, EU* and *immigration*. *tax, NHS, EU* are shared among all obtained clusters. The issue partition of 3^{rd} cluster contain *immigration* instead of *tax*.

Keywords are highly separated. There is no overlap among them. This property makes them apparent on the Figure 4.3. They are drawn as three different groups of triangles (green). First group contains keywords: *conservatives, use, Muslim, look, increasing, claims, hate, lost, price, things, chamber, gave, term*. Second group is composed of keywords: *interest, reform, give, first, want, revealed, idea, manifesto, safety, possible, rich, poll, completely, paying*. Finally, third group consists of the ones: *Theresa, pleased, sad, hall, piece, year, soon, live, improve, start, voting, history, target*.

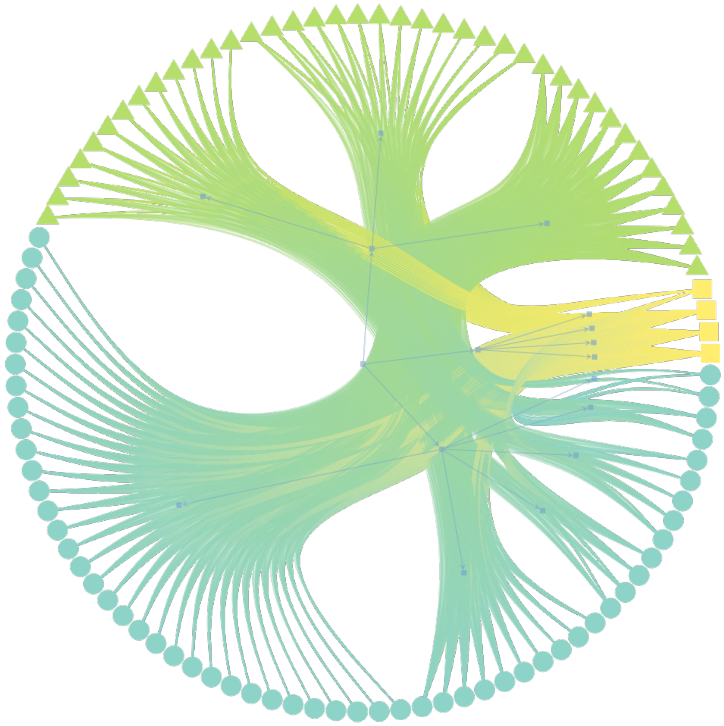


Figure 4.3: 3 Clusters in Case-2

For user nodes (blue circles), there are 3 different partitions in different clusters. The first one is composed of 49 nodes, which are mainly labors. There exists 4 nodes for conservatives, 6 ones for UKIP and 1 for SNP. Second partition consists of 14 nodes , again mainly for Labors including 2 ones for UKIP. As contrast, user partition of 3^{rd} cluster does not contain any one from Labors. 8 of 11 nodes are from UKIP, the others are from Conservatives.

When density constraints get low, the variety of opinions in a cluster increases. In test case 1, two users share 6 of 8 opinions in a cluster. In this case, this ratio is about 5 of 10 opinions. This reduction increases variety of users from different political perspective in user partition of a cluster. This level difference is good to discover which opinions are shared among different political campaigns. Moreover, level of diversity can give an opinion about how far or close different political parties are in terms of what they support.

Case 3: In this case, density constraints ϵ_p and ϵ_n remains same with case 2 as size constraints are increased (Table 4.3). These circumstances results that the

algorithm has found 2 different clusters (Figure 4.4). It is expected since it is forced to look for bigger ones. By this approach, it is aimed to find bigger user group which shares similar opinions.

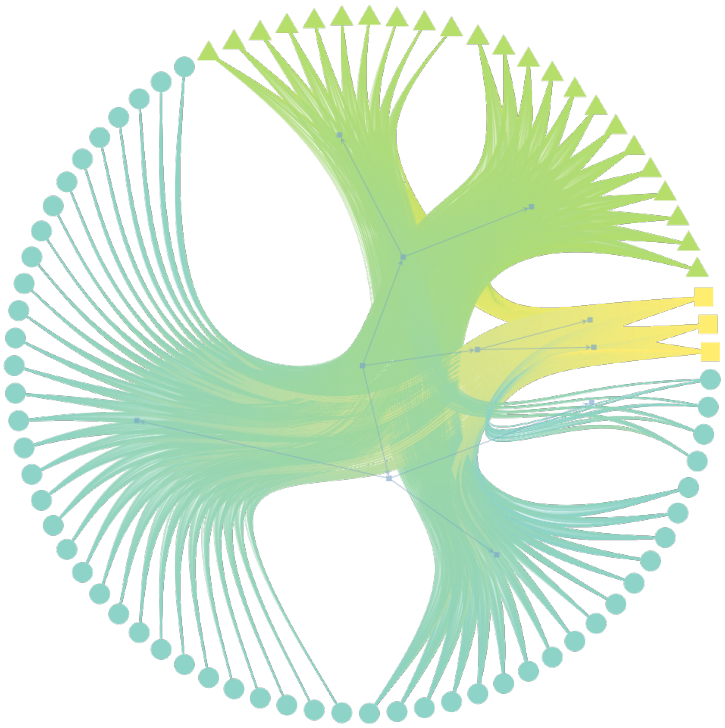


Figure 4.4: 2 Clusters in Case-3

The size of first cluster is $49 \times 3 \times 13$. Since this cluster also satisfies the constrains in case 2, it is present in the cluster list of it, too. However, the second clusters are different (Table 4.4). In case 3, the 2^{nd} cluster has 20 users, 3 issues and 10 keywords. Size constraints has just been reached. It contains 327 hyperedges, 301 of them are with positive label, 26 of them are with negative ones. The fill ratio of the whole cluster is 0.545. In case 2, the second cluster is smaller in size even though the size constraints does not effect while mining as in case 2. The size is $14 \times 3 \times 14$. Its fill ratio is lower, too. It is ~ 0.536. It contains 315 hyperedges, with 296 positive ones and 19 negative ones. Such scenario can happen, because the algorithm follows a greedy approach. At a state, the algorithm removes a least effective node. But removing this node may cause the algorithm to yield at lower local maxima. Therefore, running the algorithm several times on same input with various input parameters can help to strain it

more effectively.

Table4.4: Properties of 2^{nd} Clusters Found in Test Cases 2 and 3

Case No	Size	Fill Ratio	# Hyperedges	# +	# -	+ Ratio
2	(14,3,14)	0.536	315	296	19	0.5034
3	(20,3,10)	0.545	327	301	26	0.5017

Case 4: This case has small density constraints (ϵ_p, ϵ_n). Size constraints remain same with case 3. It has tendency to find bigger clusters but with less density of hyperedges. Sparseness will be dominant in the clusters. It is expected to have more clusters compared to case 3. As expected, it finds more clusters. There are 5 different ones (Figure 4.5).

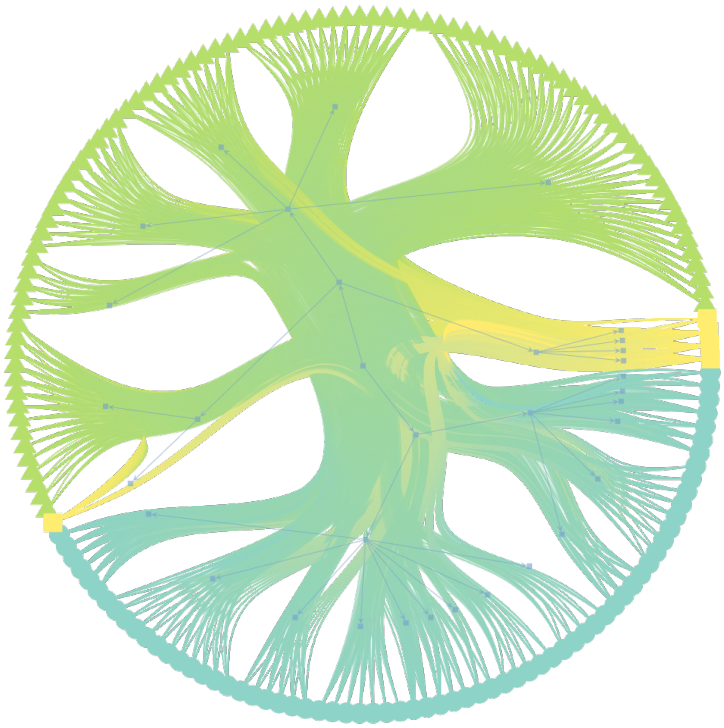


Figure 4.5: 5 Clusters in Case-4

In this case, it is more likely that obtained clusters contains higher diversity compared to ones in previous test cases. This is directly proportional to ϵ_p value. Since it is 0.3, users who shares a few opinions can be part of same

clusters. In results, there is a cluster whose users are mainly from Labours. There is also a cluster in which UKIP, the Conservatives and the Labours have equal weight.

Obstacles: On real data tests, we have faced some obstacles. The main one is that it is hard to find a data which is considerably big and dense. Since there are lots of different words in a language, users can express same feelings about an issue in various ways. This results size increase in keywords dimension which results more sparseness.

Second obstacle is hardness of extracting the real issue in a tweet which is addressed by its sentiment. For example, a user wants to satirize a politician for his/her ideas about EU. The issue of this tweet can be extracted as EU, while its sentiment will be negative. However, this opposition is not towards EU, but towards the politician. Another tweet which expresses a negative criticism about EU can be placed into a cluster of the previous tweet, because of issue and sentiment match. As a result, two tweets of same topic can be placed into same cluster. In reality, they address different issues. This situation leads to a false positive action.

4.3 Comparison

When we compare STRICLUSTER algorithm with Tucker decomposition, the basic difference is the node cover. In tucker decomposition, each node is a member of a cluster. A node can not be a part of more than one cluster. Additionally, it can not be idle, too. This is against density approach, since it increases sparseness inside clusters. Moreover, it results to ignore other potential clusters. For example, in Brexit data, one of the most popular topic is *EU*. If *EU* is covered by a cluster, then opinions of people from outside who also talk about *EU* may not be analyzed, because they are in different clusters. On the other hand, STRICLUSTER allows that a node can be a member of many or none. Test results on Brexit shows that *tax, NHS* and *EU* are shared among many clusters. Therefore, different analysis about same topic can be mined.

Furthermore, these clusters have desired quality. This feature enables us to interpret them more easily.

CHAPTER 5

CONCLUSION

In this thesis, we propose a new method, called STRICLUSTER, to find tripartite clusters of positively labeled hyperedges. The input data is composed of three dimensions. In other words, it is a hypergraph such that cardinality of each hyperedge in it is exactly 3. Each hyperedge connects three nodes from each dimension. Obtained clusters have high positivity and low negativity. These ratios are controlled by user-defined threshold values. While density of h_+ cannot be below of the threshold, density of h_- cannot exceed its own threshold. To achieve this goal, nodes in a potential cluster are removed one by one until density constraints are satisfied. On this stage, the algorithm follows a greedy approach. It determines node to remove by heuristic calculations. Least effective node is determined depending on number of h_+ by which a node is linked. The density ratio of that number helps to determine the node to omit. This way helps to cut sparseness and negativeness in the first place. As another specialty, all generated clusters satisfy some size threshold values for each dimension. This avoids to spare time and effort to look for clusters with undesired sizes. Since heuristic calculations are performed based on threshold values passed as input parameters, changing input parameters can help to discover different clusters.

Furthermore, our algorithm avoids overlap of hyperedges among generated tripartite clusters. A hyperedge of a cluster can not be contained by another one.

While mining, STRICLUSTER aims to do least number of node removal to keep cluster sizes as big as possible. Heuristic approaches are helpful for this purpose. Once size and density constraints are satisfied, node removal operation is

stopped. This behavior can be observed on test results of synthetic data. If the density of h_+ are high in the input, then number of discovered clusters reduces. It shows that big clusters are obtained. Big clusters results to reduce number of valid hyperedges quickly. Therefore, the algorithm concludes faster.

Our algorithm is willing to cover nodes as many as possible. But the nodes should be connected considerably much with positivity. Nodes which are loosely connected or negatively connected are generally out of clusters. In this manner, nodes linked by many hyperedges express stronger sentiments. They can be contained by one or more clusters, because they are more helpful on analysis. As contrast, less effective ones will contribute less for analysis. Therefore, they are more likely to be out of clusters. This behavior can be considered as filtering. In our approach, it is not necessary that a node will be a member of at least one cluster.

The other promise of our algorithm is avoiding hyperedge overlapping while allowing node overlapping. A node can be part of one or more clusters, called as node overlap. It is helpful to discover different information about nodes, since different clusters which they are part of are analyzed.

As another flexibility, since STRICLUSTER algorithm considers all dimensions with equal weights, it can be easily adapted to many real-world scenarios. In tests on Twitter data, promising results are obtained. Therefore, it may be worth to test the algorithm on other real world systems.

As a future work, the algorithm may follow more hybrid approach in node removal. To converge desired state more quickly, it can start with normalized cuts. Then, when it becomes closer to limits, it continues removals one by one. This may be helpful for data where sparseness is high. Moreover, it improves time complexity of the algorithm since the data size will decrease in a logarithmic way when cuts are applied.

Calculations of node effectiveness can be performed more efficiently. It is a complex calculation and it can be divided into sub-problems. Because sub-problems can be shared by many ones, first these sub-problems are solved, then

the calculations can use them to generate the final values. This approach is helpful in order not to repeat same problems. Consequently, the running time of the algorithm is improved.

REFERENCES

[1] B. W. Bader and T. G. Kolda. Efficient matlab computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, 2007.

[2] C. Berge. *Hypergraphs: combinatorics of finite sets*, volume 45. Elsevier, 1984.

[3] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*, volume 290. Citeseer, 1976.

[4] G. Chartrand, L. Lesniak, and P. Zhang. *Graphs & digraphs*, volume 39, page 16. CRC Press, 2010.

[5] H. Cheng, P.-N. Tan, J. Sticklen, and W. F. Punch. Recommendation via query centered random walk on k-partite graph. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 457–462. IEEE, 2007.

[6] Y. Cheng and G. M. Church. Biclustering of expression data. In *Ismb*, volume 8, pages 93–103, 2000.

[7] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[8] M. Dawande, P. Keskinocak, J. M. Swaminathan, and S. Tayur. On bipartite and multipartite clique problems. *Journal of Algorithms*, 41(2):388–403, 2001.

[9] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–274. ACM, 2001.

[10] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine learning*, 42(1-2):143–175, 2001.

[11] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[12] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998.

[13] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[14] N. Grinberg, P. A. Dow, L. A. Adamic, and M. Naaman. Changes in engagement before and after posting to facebook. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 564–574. ACM, 2016.

[15] L. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE transactions on computer-aided design of integrated circuits and systems*, 11(9):1074–1085, 1992.

[16] E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information processing letters*, 76(4):175–181, 2000.

[17] S. B. Hobolt. The brexit vote: a divided nation, a divided continent. *Journal of European Public Policy*, 23(9):1259–1277, 2016.

[18] J.-J. Huang, G.-H. Tzeng, and C.-S. Ong. Marketing segmentation using support vector clustering. *Expert systems with applications*, 32(2):313–317, 2007.

[19] T. G. Kolda and J. Sun. Scalable tensor decompositions for multi-aspect data mining. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 363–372. IEEE, 2008.

[20] N. Leavitt. Recommendation technology: Will it boost e-commerce? *Computer*, 39(5):13–16, 2006.

[21] N. Li and G. Chen. Analysis of a location-based social network. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 4, pages 263–270. Ieee, 2009.

[22] Y.-R. Lin, J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher. Metafac: community discovery via relational hypergraph factorization. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 527–536. ACM, 2009.

[23] X. Liu and T. Murata. Advanced modularity-specialized label propagation algorithm for detecting communities in networks. *Physica A: Statistical Mechanics and its Applications*, 389(7):1493–1500, 2010.

[24] X. Liu and T. Murata. Detecting communities in tripartite hypergraphs. *arXiv preprint arXiv:1011.1043*, 2010.

[25] B. Long, Z. M. Zhang, X. Wu, and P. S. Yu. Spectral clustering for multi-type relational data. In *Proceedings of the 23rd international conference on Machine learning*, pages 585–592. ACM, 2006.

[26] C. Lu, X. Chen, and E. Park. Exploit the tripartite network of social tagging for web clustering. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1545–1548. ACM, 2009.

[27] A. McGregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *International Workshop on Passive and Active Network Measurement*, pages 205–214. Springer, 2004.

[28] M. E. Newman. Communities, modules and large-scale structure in networks. *Nature Physics*, 8(1):25–31, 2012.

[29] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[30] T. N. Pappas. An adaptive clustering algorithm for image segmentation. *IEEE Transactions on signal processing*, 40(4):901–914, 1992.

[31] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[32] M. Wieling and J. Nerbonne. Bipartite spectral graph partitioning to co-cluster varieties and sound correspondences in dialectology. In *Proceedings of the 2009 Workshop on Graph-based Methods for Natural Language Processing*, pages 14–22. Association for Computational Linguistics, 2009.

[33] C.-m. A. Yeung, N. Gibbins, and N. Shadbolt. Tag meaning disambiguation through analysis of tripartite structure of folksonomies. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Workshops*, pages 3–6. IEEE Computer Society, 2007.

[34] K. Y. Yeung and W. L. Ruzzo. Principal component analysis for clustering gene expression data. *Bioinformatics*, 17(9):763–774, 2001.

[35] L. Zhao and M. J. Zaki. Tricluster: an effective algorithm for mining coherent clusters in 3d microarray data. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 694–705. ACM, 2005.

[36] L. Zhu, A. Galstyan, J. Cheng, and K. Lerman. Tripartite graph clustering for dynamic sentiment analysis on social media. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1531–1542. ACM, 2014.