# Multi-layer bipartite structural features to analyze YouTube Social Network

*Maria Oikonomidou*

Thesis submitted in partial fulfillment of the requirements for the

*Masters' of Science degree in Computer Science and Engineering*

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Associate Prof. *Polyvios Pratikakis*

---

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Greek GSRT through the project ETAK, with project ID T1EDK-01800

University of Crete
Computer Science Department

**Title: Multi-layer bipartite structural features to analyze YouTube Social Network**

Thesis submitted by
**Maria Oikonomidou**
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: ______________________
Maria Oikonomidou

Committee approvals: ______________________
Polyvios Pratikakis
Associate Professor, Thesis Supervisor

______________________
Ioannis Tollis
Professor, Committee Member

______________________
Charalampos Saridakis
Associate Professor, Committee Member

Departmental approval: ______________________
Polyvios Pratikakis
Associate Professor, Director of Graduate Studies

Heraklion, February 2022

# Multi-layer bipartite structural features to analyze YouTube Social Network


## Abstract

This work investigates interactions on YouTube, concerning predicting missing or unseen interactions on multi-layer bipartite networks. More precisely, given a set of own interactions between YouTube users and videos, we measure how accurately we can predict comment interactions. We propose structural bipartite features, which enhance the performance of simple prediction models, to find missing or unseen links. Experimental validation of the proposed approach is carried out on multi-layer networks formed on YouTube. We have crawled an extensive dataset of YouTube videos, the channels that own them, and the authors of their comments. Using a machine learning framework, we find that we can predict future and unseen comment interactions on YouTube videos with precision 99%. We also show that to predict a day's comment interactions it suffices to account network information generated 1 day prior. Our set-up is implemented on the MapReduce model. We propose two MapReduce algorithms, one that counts the bitruss number of an edge and one that clusters edges into blooms in a bipartite network.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In this study, we investigate the comment interaction on the YouTube social network. The goal is to predict which YouTube user is commenting on what video. We approach this comment prediction task as an edge prediction problem for multi-layer bipartite graphs. We propose multi-layer structural features and compare their predictive power over simple structural features, on the YouTube bipartite networks.

YouTube is an online video sharing and social media platform. It is the second most visited website, with more than one billion monthly users [1]. YouTube users can watch, like, share, comment, put into a playlist, and upload their videos. Prior research studies on the social media platform have focused primarily on analyzing existing content and providing statistical insights for the platform [9, 21, 27, 28]. There is very little work related to predicting interactions on the platform solely.

The task of predicting interactions fits naturally in a graph framework. For that reason, we use bipartite graphs (bigraphs) to model relations between users and videos on YouTube. More precisely, we represent in a bigraph the relation ‘own’, which is *user owns video* and the relation ‘comment’, which is *user comments video*. Following, with the use of the topological features emerging from the ‘own’ and ‘comment’ graph, we answer the question ‘which user is going to comment on what video’.

To make comment predictions on YouTube, we translate the problem to a link prediction task. Given a network at a specific period of time, the link prediction problem is considered as a task of discovering unseen links or predicting new ones that will occur in a future time. In this work, we use the link prediction as a binary classification problem, where first-order structures such as the node degree, edge existence, and the high-order structures such as the butterfly, the bloom, and the multi-layer topological structures are used as features.

A ***butterfly***, also known as ***rectangle*** [2, 19, 20, 25, 24], is a complete 2 x 2 biclique and is considered as an analogue of a triangle in unipartite graphs. Figure 1.1 shows a user-video bipartite network with 2 butterflies formed, marked in dotted lines. We also use the ***bloom*** [26] as a structural feature to infer information about the bipartite network. Blooms can be considered clusters of butterflies. An example of bloom construction out of a bipartite graph is in figure 3.2. As high-order structural features, we also consider the topological features emerging when combining the 'own' and 'comment' interactions. These two YouTube interactions characterize nodes and their connections on two different relations. This setting is also known as ***multi-layer network***, and we take it into account to make more sophisticated predictions.

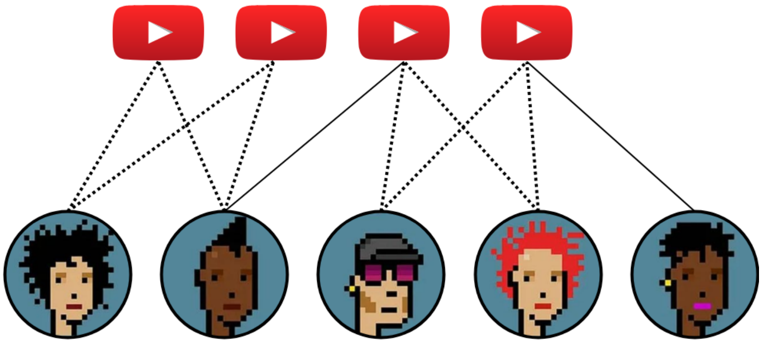


Figure 1.1: A user-video bipartite network with 2 butterflies (dotted edges).

Overall, this study makes the following contributions:

- We compare the predictive power of first-order to high-order structures on the missing link task on a multi-layer bipartite network.
- We compare the predictive power of first-order to high-order structures on the formation of future links on a multi-layer bipartite network.
- We show that to predict a day's comment interactions suffices to consider data generated one day prior.
- We implement a MapReduce algorithm that counts the bitruss number of an edge in a bipartite network.
- We implement a MapReduce algorithm that clusters edges of a bipartite network into blooms.

# Chapter 2

# Related work

Given a network at a specific time, the link prediction problem is a task of discovering unseen links or predicting new ones that will occur in a future time. The link prediction has been extensively studied in unipartite and bipartite networks [17, 4, 18, 15, 22, 6, 5, 16, 13]. Al Hasan et al. [3] introduce the link prediction problem as a binary classification task, where various similarity metrics and data outside the graph topology scope were used as features. The same supervised classification approach has also been used by [10, 23, 7].

In unipartite graphs, link prediction algorithms make the assumptions of Triangle closure and Clustering [14], which cannot be in use on bipartite graphs. Technically, unipartite link prediction algorithms can apply to bipartite graphs, but they will not perform well because they are based on the triangle closure. For that reason, the link prediction algorithms need to be fine-tuned to match the structure of bipartite graphs.

In this study, we approach the link prediction problem as a binary classification task. We adopt multi-layer higher-order topological features that have application only on bipartite graphs. In contrast, authors of [6, 8] adapt some topological measures used in unipartite graphs for predicting links in bipartite graphs. They also transform bipartite graphs into unipartite and perform link prediction. The work of Allali et al. [5] propose a link prediction method that is based on the notion that if two nodes have a common neighbor in the graph, they will probably acquire more in the future.

We expand the idea of the triangle closure and build upon the analog structure for bipartite graphs, which is the butterfly. A butterfly (also known as rectangle), is a complete 2 x 2 biclique [24, 19, 25]. Besides the butterfly structure, we use the notion of the *Bloom* structure and *bitruss* number [26] of a link for our bipartite graph embedding. To enhance the predictive features, we also leverage structural features emerging when considering multi-layer graphs.

Multi-layer networks characterize multiple types of interactions not possible to represent by using a traditional monolayer network approach. Figure 2.1 shows an example of a multi-layer bipartite network. Each layer represents a bipartite network, whose vertices are divided into two disjoint and independent sets, such that every edge connects a vertex from one set to the other [11]. In figure 2.1, there is a set with circle vertices and a set with square vertices represented on two networks, on Layer 1 and on Layer 2. For example, Layer 1 can represent the 'own' relations and Layer 2 can represent the 'comment' relation.

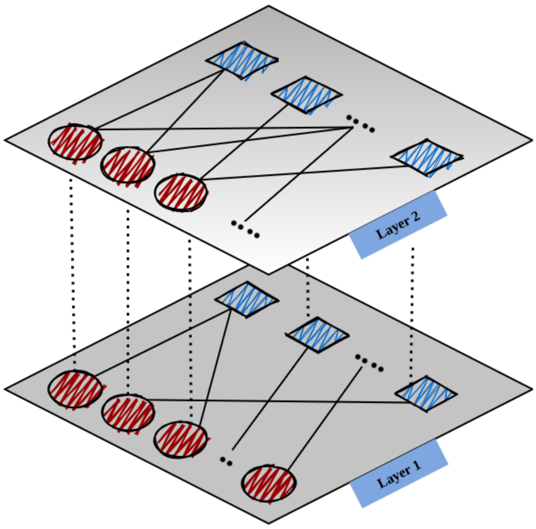



Figure 2.1: Multi-layer bipartite graph with 2 layers.

Close to our work lies the framework proposed in [22], where they study user interactions on Twitter. They use multi-layer directed networks and show that features involving triads turn out to be important for accurate predictions. There is little work regarding multi-layer bipartite link prediction. The study of [12] propose a community detection-base measures for link prediction.To the best of our knowledge, we are the first to address the multi-layer bipartite link prediction while taking into account multi-layer features on YouTube. We create a graph embedding with first-order and high-order structural features, which consists of multi-layer butterfly motif structures on the YouTube comment and own network.

# Chapter 3

# Design

In this section, we describe the set-up for the multi-layer link prediction in bipartite graphs, to predict the comment action on YouTube. We model the bipartite link prediction problem as a binary classification task, where each data point corresponds to a pair of vertices in the network. Vertices are divided into two disjoint and independent sets U and V, such that they compose an edge connecting a vertex in U to one in V.

The link prediction problem corresponds to two tasks, one to predict future links and one to predict missing links in a network. In our set-up we explore both prediction tasks. We need to create a graph embedding to perform both link prediction tasks. For this study, the graph embedding for these tasks is the same and is a vector that contains structural information for every edge/link in the network. Specifically, the vector contains first-order and high-order structural features of the multi-layer bipartite network. The first-order structural features are the degree of a node and the existence of an edge in the different layers. The high-order structural features we use in this study emerge from the butterfly motif and are *the blooms, the bitruss number, and the multi-layer butterflies.*

## 1 Definitions

Our problem is defined as an undirected bipartite graph $G(U, V, E)$, where $U(G)$ denotes the set of vertices in partition U and $V(G)$ denotes the set of vertices in partition V with $U(G) \cap V(G) = \varnothing$, and $E(G) \subseteq U(G) \times V(G)$ denotes the edge set. An edge between two vertices a and b in G is denoted as $(a, b)$ or $(b, a)$.

**Butterfly (Rectangle):** Given a bipartite graph G (single layer) and four vertices $(a, b, \in V(G)$ and $c, d \in U(G))$, a butterfly induced by the vertices a, b, c, d is a (2,2)-biclique of G; that is, a and b are both connected to c and d, respectively, by edges $(a, c), (a, d), (b, c), (b, d) \in E(G)$. For example in figure 3.1 there are 2 butterflies. One butterfly is supported by the edges((U1,V1), (U1,V2), (U2,V1), (U2,V2)) and the other is supported by the edges((U2,V2), (U2,V3), (U3,V2), (U3,V3)).

**Bitruss number:** Given a bipartite graph G, the bitruss number of an edge e, denoted as b(e), is a number k that indicates how many butterflies e supports. Figure 3.1 shows that the dotted edge(U2,V2) has bitruss number equal to 2, because this edge supports 2 butterflies, the solid edges((U1,V1), (U1,V2), (U2,V1), (U2,V3), (U3,V2), (U3,V3)) have bitruss number = 1, and the dashed edges((U4,V3), (U4,V4), (U4,V5)) have bitruss number = 0 because they do not support any butterfly.

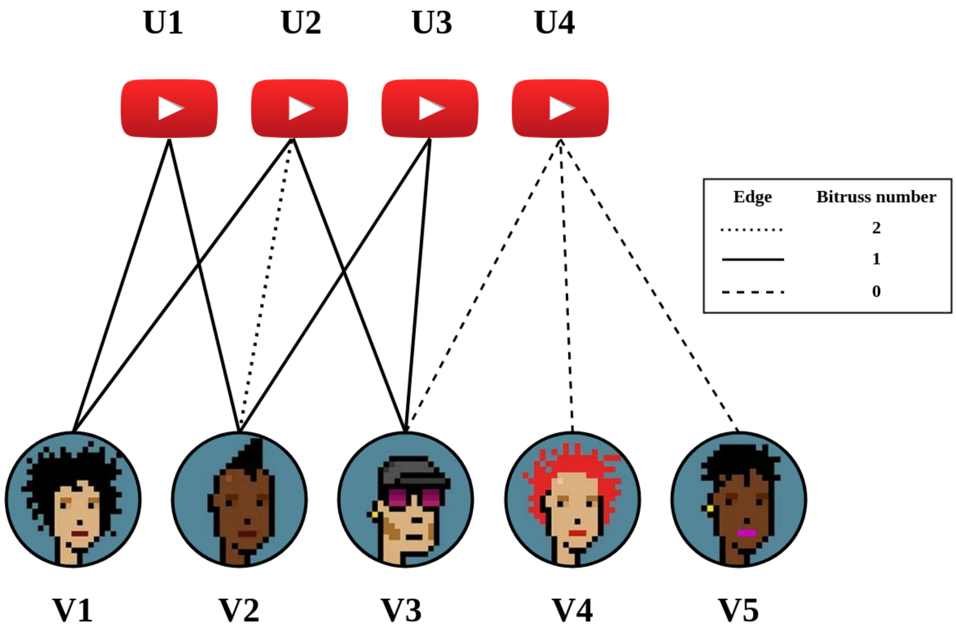



Figure 3.1: A bipartite network with 2 butterflies and the bitruss count of each edge.

**Bloom:** Given a bipartite graph G, a bloom [26] is a cluster of butterflies. Blooms concentrate a set of edges that support butterflies that share at least one edge. Each edge in G can only be part of one bloom. The bipartite network in figure 3.2 has only one bloom with id V1U1. The bloom V1U1 is supported by the edges of butterflies made by the vertices [U1, U2, V1, V2] and [U2, U3, V2, V3].

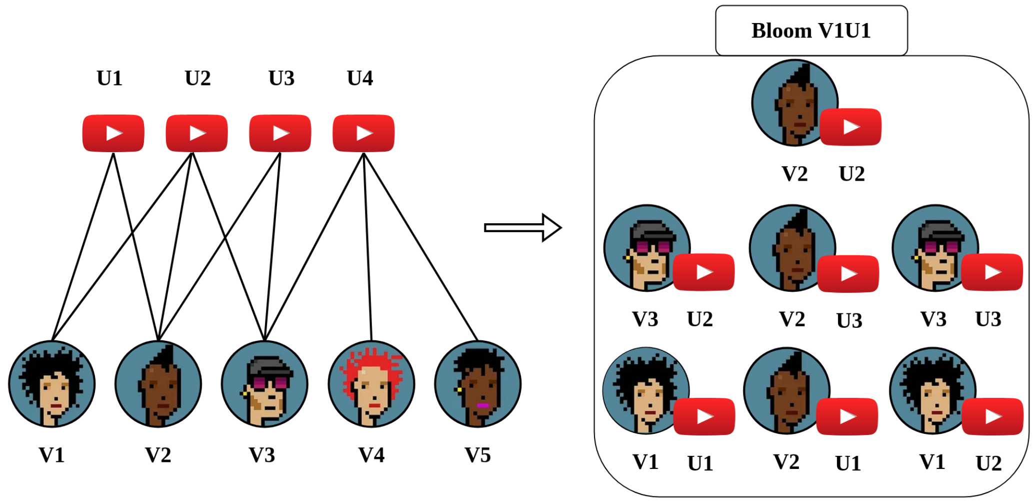



Figure 3.2: Bloom construction out of a bipartite network.

**Multi-layer Butterfly:** Given two bipartite graphs $G_1$ (Layer 1), $G_2$ (Layer 2) and 4 vertices connected with 4 edges. A multi-layer butterfly extends the criteria of the single-layer butterfly and has two properties regarding the butterfly edges. First property is to have 1 edge at one layer and the remaining 3 at the other layer, and the second property is to have 2 edges at one layer and the remaining 2 at the other layer. In total, we define six structural types of multi-layer butterflies shown in figure 3.3. We set Layer 1 ($G_1$) to be the relation 'own', marked in dotted lines and Layer 2 ($G_2$) to be the relation 'comment', marked in solid lines. For example the Type 3 multi-layer butterfly structure from the figure 3.3 indicates that the two users own a video and that these two users have commented on each other's video.

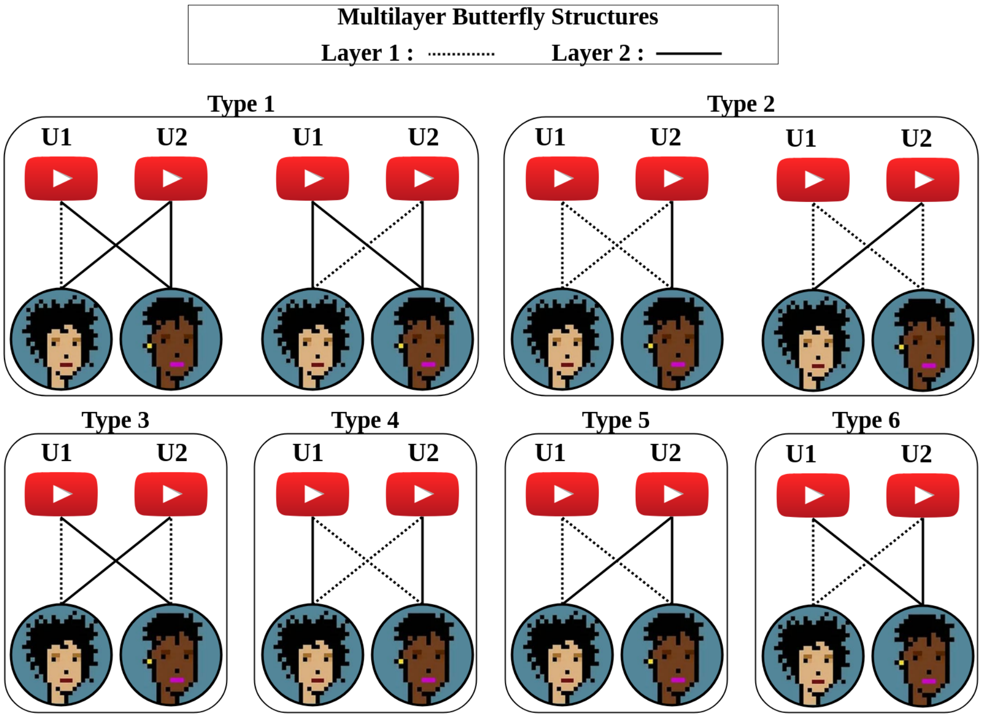



Figure 3.3: The 6 structural types of multi-layer butterflies.

## 2 MapReduce Algorithms

For the bipartite link prediction implementation, we use the MapReduce programming model to scale out the problem. More specifically, we use Apache Spark, which uses the MapReduce distributed computing framework as its foundation. Spark is a multi-language engine for executing data engineering, data science, and machine learning on a single-node machine or clusters.

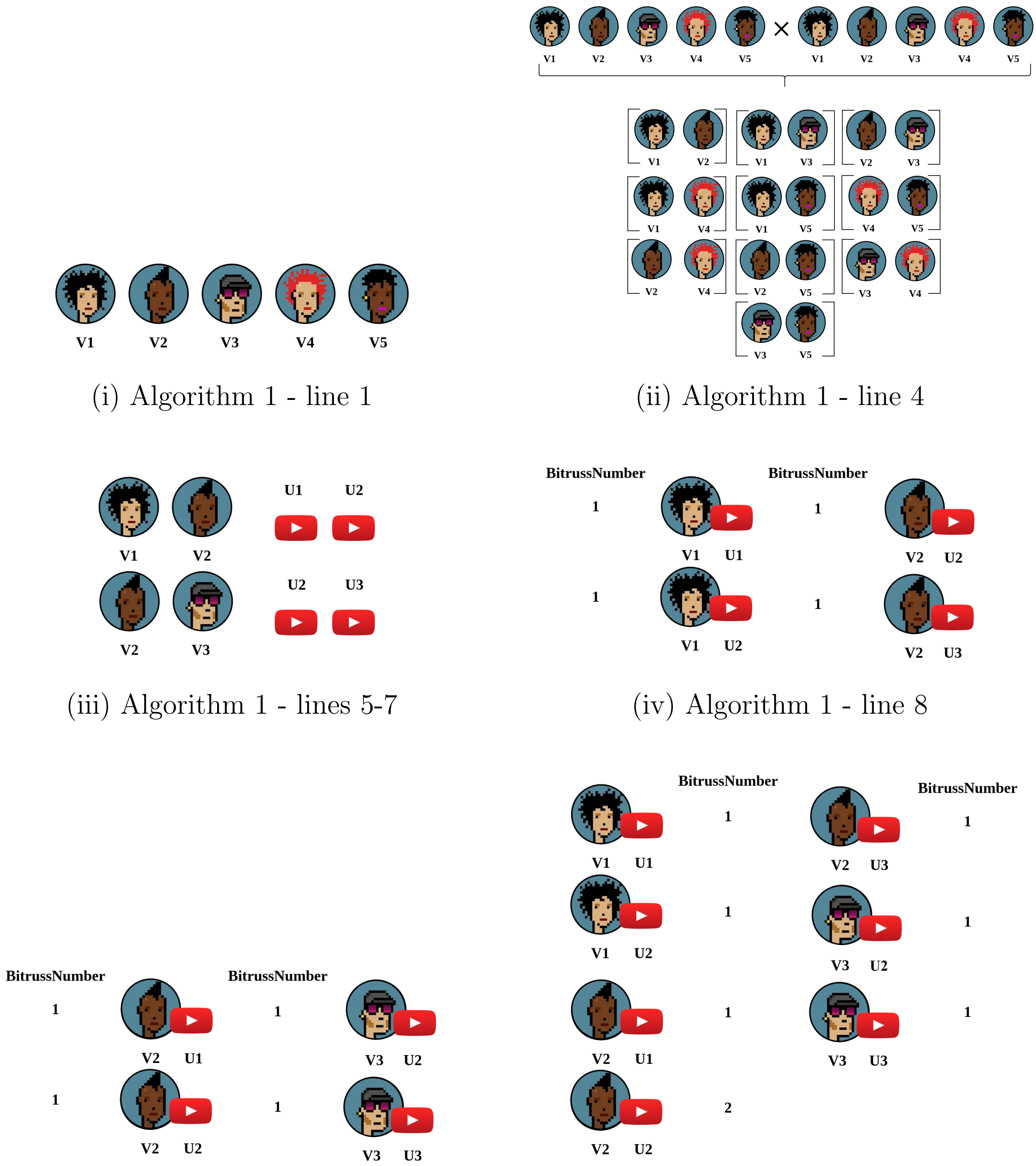
Following. we present the MapReduce pseudocode to describe the algorithms we use for the Edge Bitruss Count (algorithm 1) and the Edge Bloom Identification (algorithm 2) in a bipartite graph.

Given a bipartite network, the **Edge Bitruss Count** Algorithm 1 computes the bitruss number of an edge $b(e)$, for $b(e) \geq 1$. The bitruss number of an edge represents the number of butterflies the edge supports. The algorithm 1 returns a list of edges with their bitruss number.

For example, given as input to the algorithm 1 the Graph of figure 3.1, in **line 1** we gather all nodes from set V of the graph G, so we get $src\_set \leftarrow$ [V1,V2,V3,V4,V5], also shown in figure 3.4(i). In **line 2**, we declare a function named N(n), which for every given node n it returns a list with its neighbors. In **line 3**, we declare the function count_butterflies(list), that given a list it returns its size minus 1. This is because 2 source nodes can have butterfly support equal to the number of their common neighbors minus 1. In **line 4**, we performe a cartesian product to all the node of set V of src_set where $V_i \neq V_j$, so *src_cartesian* $\leftarrow$ [(V1,V2), (V1,V3), (V1,V4), (V1,V5), (V2,V3), (V2,V4), (V2,V5), (V3,V4), (V3,V5), (V4,V5)], also shown in figure 3.4(ii).

In **lines 5-7**, we calculate the potential butterflies for each pair created in src_cartesian, if they have more or equal to 2 neighbors. This condition is because two source nodes must have at least two common neighbors to form a butterfly. So $butterflies \leftarrow$ [((V1,V2),(U1,U2)), ((V2,V3),(U2,U3))], also shown in figure 3.4(iii). In **lines 8-11**, we select the first source node from the pair of source nodes, with all the destination nodes in the list of common neighbors, so as to construct an edge, and compute the bitruss number of them with the function count_butterflies. So for our example we get $bitruss\_src_1 \leftarrow$ [(1,(V1,U1)), (1,(V1,U2)), (1,(V2,U2)), (1,(V2,U3))], shown in figure 3.4(iv). Next we do the same for the second source node and we get $bitruss\_src_2 \leftarrow$ [(1,(V2,U1)), (1,(V2,U2)), (1,(V3,U2)), (1,(V3,U3)], also shown in figure 3.4(v). Following in **lines 12-13**, we join the $bitruss\_src_1$ and $bitruss\_src_2$ and we get a set that has a pair of source and destination node, which is an edge of G and its bitruss number. So $bitruss \leftarrow$ [((V1,U1),1), ((V1,U2),1), ((V2,U1),1), ((V2,U2),1), ((V2,U3),1), ((V2,U2),1), ((V3,U2),1), ((V3,U3),1)]. Last in **line 14**, for each edge in the bitruss set we sum (reduce) their bitruss number and we get the result set that contains edges and their total bitruss number. For our example in this step we get $edge\_bitruss \leftarrow$ [((V1,U1),1), ((V1,U2),1), ((V2,U1),1), ((V2,U2),2), ((V2,U3),1), ((V3,U2),1), ((V3,U3),1)], as shown in figure 3.4(vi).

Following we descrive the space complexity of the **Edge Bitruss Count** Algorithm 1. For input we have a graph G=(U,V,E), where V are the nodes in set V, U are the nodes in set V and E are the edges between nodes of set U and set V. In **line 1** we have $|src\_set| = |V|$. In **line 2** we declare a function N that gets a node n and returns its neighbors. In our case we give as input the nodes of set V, so we have $N(n) \subseteq U$. We also have that $\sum_{n \in src\_set} |N(n)| = |E|$. In **line 4** we get $|src\_cartesian| = |V|^2 - |V| = O(|V|^2)$. For **lines 5-7**, we have $|butterflies| \leq |src\_cartesian| + O(|V|^2)$. We get $bitruss\_src_1 \leq O(|V|^2 * |E|)$, in **lines 8-9**. We have the same complexity for $bitruss\_src_2 \leq O(|V|^2 * |E|)$, in **lines 10-11**. In **lines 12-13** we get $bitruss \leq O(|V|^2) * |E| + O(|V|^2) * |E|$, since is the union of $bitruss\_src_1$ and $bitruss\_src_2$. Last step in **line 14** we have $edge\_bitruss \leq |E|$.




(i) Algorithm 1 - line 1 (ii) Algorithm 1 - line 4 (iii) Algorithm 1 - lines 5-7 (iv) Algorithm 1 - line 8 (v) Algorithm 1 - lines 10 (vi) Algorithm 1 - line 14

Figure 3.4: Illustration of the Algorithm 1 with input the Graph of figure 3.1.

**Algorithm 1** Edge Bitruss Count

**Input:** $Graph\, G = (U, V, E)$

1: $src\_set \leftarrow V(G)$ */* V(G) are the source nodes */*

2: $N(n)$ */* Returns the neighbor set of node n */*

3: $count\_butterflies(list)$ */* Gets a list of destination nodes and returns list.size-1 */* */* The number of butterflies 2 source nodes support, is equal to the number of their common neighbors - 1 */*

4: $src\_cartesian \leftarrow src\_set \times src\_set \; where \; V_i \neq V_j$

5: $butterflies \leftarrow (\forall(src_1, src_2) \in src\_cartesian).map((src_1, src_2),$

6: $N(src_1) \cap N(src_2)).$

7: $filter(N(src_1) \cap N(src_2) \geq 2)$ */* To compose a butterfly 2 source nodes must have at least 2 common neighbors */*

8: $bitruss\_src_1 \leftarrow butterflies.map(count\_butterflies(N(src_1) \cap N(src_2)) :$

9: $BitrussNumber, src_1 \times N(src_1) \cap N(src_2))$

10: $bitruss\_src_2 \leftarrow butterflies.map(count\_butterflies(N(src_1) \cap N(src_2)) :$

11: $BitrussNumber, src_2 \times N(src_1) \cap N(src_2))$

12: $bitruss \leftarrow bitruss\_src_1 \cup bitruss\_src_2.$

13: $map((src, dst) : Edge, BitrussNumber)$

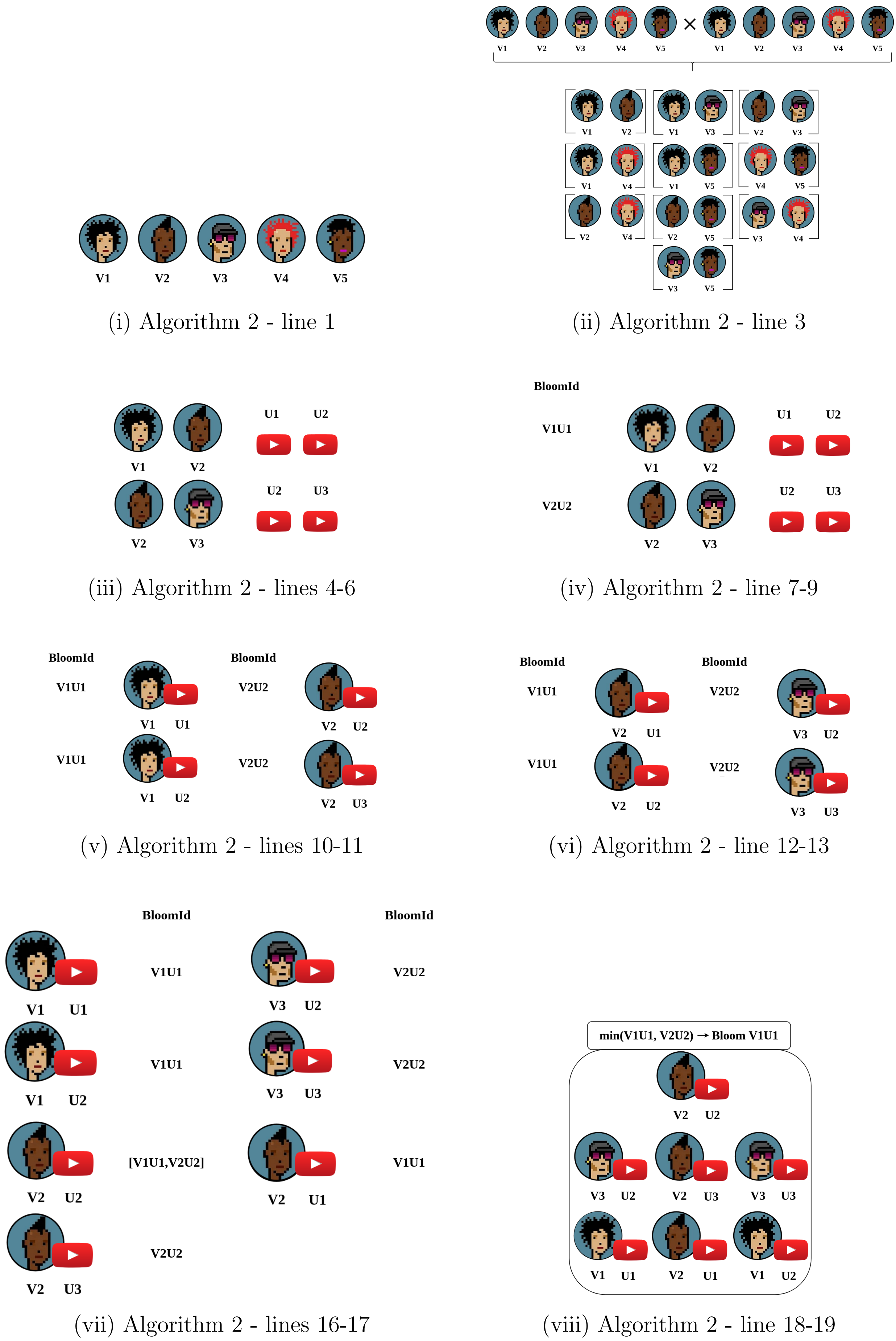
14: $edge\_bitruss \leftarrow bitruss.reduce(Edge, BitrussNumber)$

15: $return\; edge\_bitruss$

Given a bipartite network, the **Bloom Identification** Algorithm 2 finds and constructs the blooms that exist in the network. Blooms concentrate a set of edges that construct butterflies that share at least one edge. We give an id to the bloom based on the vertices it concentrates.

For example, given as input to the algorithm 2 the Graph of figure 3.2, in **line 1** we gather all nodes from set V of the graph G, so we get $src\_set \leftarrow$ [V1,V2,V3,V4,V5], also shown in figure 3.5(i). In **line 2**, we declare a function named N(n), which for every given node n it returns a list with its neighbors. In **line 3**, we performe a cartesian product to all the node of set V of src_set where $V_i \neq V_j$, so $src\_cartesian \leftarrow$ [(V1,V2), (V1,V3), (V1,V4), (V1,V5), (V2,V3), (V2,V4), (V2,V5), (V3,V4), (V3,V5), (V4,V5)], also shown in figure 3.5(ii). In **lines 4-6**, we calculate the potential butterflies for each pair of src_cartesian, if they have more or equal to 2 neighbors. This condition is because two source nodes must have at least two common neighbors to form a butterfly. So $butterflies \leftarrow$ [((V1,V2),(U1,U2)), ((V2,V3),(U2,U3))], also shown in figure 3.5(iii). In **lines 7-9**, we create the potential blooms. We construct a potential bloom id by selecting the minimum id between the two source node ids and the minimum id from the neighbor' nodes of these source nodes and append it to the set of butterflies. So we have $potential\_bloomids \leftarrow$ [(V1U1,(V1,V2),(U1,U2)), (V2U2,(V2,V3),(U2,U3))], also shown in figure 3.5(iv). Following in **lines 10-13**, we select the bloom id and the source nodes with all the destination nodes in the list of common neighbors and create all the potential edges with the bloom id they are given in the previous step. First we take the first node from the two source node, so we get $edge\_src_1\_bloomids \leftarrow$ [(V1U1,(V1,U1)), (V1U1,(V1,U2)), (V2U2,(V2,U2)) (V2U2,(V2,U3))], which is shown in figure 3.5(v). Next we do the same for the second node from the two source node pair and we get $edge\_src_2\_bloomids \leftarrow$ [(V1U1,(V2,U1)), (V1U1,(V2,U2)), (V2U2,(V3,U2)), (V2U2,(V3,U3))], shown in figure 3.5(vi). In **lines 14-15**, we take $edge\_src_1\_bloomids$ and $edge\_src_2\_bloomids$, and construct the union of these two and as a result we get a set that has a bloom id and an edge of G. So in our example we get $edge\_bloomids \leftarrow$ [((V1,U1),V1U1), ((V1,U2),V1U1), ((V2,U2),V2U2), ((V2,U3),V2U2), ((V2,U2),V1U1), ((V3,U2),V2U2), ((V3,U3),V2U2),((V2,U1),V1U1)]. In the $edge\_src_2\_bloomids$ set in our example, the edge (V2,U2) has two bloom ids assigned the V1U1 and the V2U2. To distinguish which bloom id to assign to similar cases we do the following steps. In lines **lines 16-17** we group by edge the $edge\_src_2\_bloomids$ set. We have $edge\_belong\_blooms \leftarrow$ [((V1,U1),[V1U1]), ((V1,U2),[V1U1]), ((V2,U2),[V1U1,V2U2]), ((V2,U3),[V2U2]), ((V3,U2),[V2U2]), ((V3,U3),[V2U2]), ((V2,U1),[V1U1])], as shown in figure 3.5(vii) In **lines 18-19**, we now as last step we group by the bloom ids of the $edge\_belong\_blooms$ set. From the result of grouping we select from the bloom ids set the minimum id. The minimum id is now the unique identifier of the bloom and the set of edges that belong to it. So we have $blooms \leftarrow$ [V1U1,((V2,U2),(V3,U2), (V2,U3), (V3,U3),(V1,U1), (V2,U1), (V1,U2))], as shown in in figure 3.5(viii).

Following we descrive the space complexity of the **Bloom Identification** Algorithm 2. For input we have a graph G=(U,V,E), where V is the nodes in set V, U is the set of nodes in set V and E are the edges between the nodes of U set and V set. In **line 1** we have $|src\_set| = |V|$. In **line 2** we declare a function N that gets a node n and returns its neighbors. In our case we give as input the nodes of set V, so we have $N(n) \subseteq U$. We also have that $\sum_{n \in src\_set} |N(n)| = |E|$. In **line 3** we get $|src\_cartesian| = |V|^2 - |V| = O(|V|^2)$. For **lines 4-6**, we have $|butterflies| \leq |src\_cartesian| + O(|V|^2)$. We get $edge\_src_1\_bloomids \leq O(|V|^2 * |E|)$, in **lines 10-11**. We also have $edge\_src_2\_bloomids \leq O(|V|^2 * |E|)$, in **lines 12-13**. In **lines 14-15** we get $edge\_bloom \leq O(|V|^2) * |E| + O(|V|^2) * |E|$, since is the union of $edge\_src_1\_bloomids$ and $edge\_src_2\_bloomids$. Next in **lines 16-17** we have $edge\_belong\_blooms \leq |E|$. Last step in **line 18-19** we have $blooms \leq$ |butterflies in the G|.




(i) Algorithm 2 - line 1

(ii) Algorithm 2 - line 3

(iii) Algorithm 2 - lines 4-6

(iv) Algorithm 2 - line 7-9

(v) Algorithm 2 - lines 10-11

(vi) Algorithm 2 - line 12-13

(vii) Algorithm 2 - lines 16-17

(viii) Algorithm 2 - line 18-19

Figure 3.5: Illustration of the Algorithm 2 with input the Graph of figure 3.2.

**Algorithm 2** Bloom Identification

**Input:** $Graph = (U, V, E)$

$src\_set \leftarrow V(G)$ */* V(G) are the source nodes */*
$N(n)$ */* Returns the neighbor set of node n */*

$src\_cartesian \leftarrow src\_set \times src\_set$ *where* $V_i \neq V_j$

$butterflies \leftarrow (\forall(src_1, src_2) \in src\_cartesian).map((src_1, src_2),$
$N(src_1) \cap N(src_2)).$
$filter(N(src_1) \cap N(src_2) \geq 2)$
*/* To compose a butterfly 2 source nodes must have at least 2 common neighbors */*
$potential\_bloomids \leftarrow butterflies.map(([src_1, src_2].min,$
$(N(src_1) \cap N(src_2)).min) : BloomId,$
$src_1, src_2, (N(src_1) \cap N(src_2)))$

$edge\_src_1\_bloomids \leftarrow potential\_bloomids.map(BloomId,$
$(src_1 \times N(src_1) \cap N(src_2)).combinations(2) : Edge)$

$edge\_src_2\_bloomids \leftarrow potential\_bloomids.map(BloomId,$
$(src_2 \times N(src_1) \cap N(src_2)).combinations(2) : Edge)$

$edge\_bloomids \leftarrow (edge\_src_1\_bloomids \cup edge\_src_2\_bloomids).$
$map(BloomId, Edge)$

$edge\_belong\_blooms \leftarrow edge\_bloomids.groupBy(Edge).$
$map(Edge, Set(BloomId) : BloomIds)$

$blooms \leftarrow edge\_belong\_blooms.groupBy(BloomIds).$
$map(BloomIds.min, Set(Edges))$

*return blooms*

# 3 Bipartite Graph Features

In this section, we describe the 34 bipartite graph features we use to perform the multi-layer link prediction. We perform two tasks of the link prediction in a bipartite network. The first task is to predict future interactions, and the second is to predict unseen interactions. Since link prediction is a typical binary classification task, we use a simple logistic regression model to make predictions.

For both tasks, we create a graph embedding that is composed of the 34 bipartite graph features described in table 3.1. Besides the simple first-order structural features from no 1 to 10 on table 3.1, we propose the high-order structural features of the multi-layer bipartite network that are from no 11 to 28 and features emerging from the butterfly motif that are from no 29 to 34 in the table. To the best of our knowledge, these high-order features have not been used before in the literature for the bipartite link prediction problem.

In our set up given 2 graphs, Layer 1 ($G_1$) and Layer 2 ($G_2$), we aim to make link predictions for both tasks on Layer 2. For every edge in Layer 1 and Layer 2 we calculate the features in table 3.1 and compose the multi-layer graph embedding. The multi-layer graph embedding is given as input to the Logistic regression model to make predictions for both tasks.

| Structural Features | no. | Feature Name | Value | Description |
|---|---|---|---|---|
| | 1 | EdgeinLayer1 | 1 \| 0 | Edge exists in Layer 1 |
| | 2 | EdgeinLayer2 | 1 \| 0 | Edge exists in Layer 2 |
| | 3 | SrcDegreeLayer1 | N | Degree of source node in Layer 1 |
| | 4 | SrcinLayer1 | 1 \| 0 | Source node exists in Layer 1 |
| first-order | 5 | SrcDegreeLayer2 | N | Degree of source node in Layer 2 |
| | 6 | SrcinLayer2 | 1 \| 0 | Source node exists in Layer 2 |
| | 7 | DstDegreeLayer1 | N | Degree of destination node in Layer 1 |
| | 8 | DstinLayer1 | 1 \| 0 | Destination node exists in Layer 1 |
| | 9 | DstDegreeLayer2 | N | Degree of destination node in Layer 2 |
| | 10 | DstinLayer2 | 1 \| 0 | Destination node exists in Layer 1 |
| | 11 | MultiButType1 | N | Number of times the Edge is part of Multi-Layer Butterfly Structure Type 1 |
| | 12 | MultiButType2 | N | Number of times the Edge is part of Multi-Layer Butterfly Structure Type 2 |
| | 13 | MultiButType3 | N | Number of times the Edge is part of Multi-Layer Butterfly Structure Type 3 |
| | 14 | MultiButType4 | N | Number of times the Edge is part of Multi-Layer Butterfly Structure Type 4 |
| | 15 | MultiButType5 | N | Number of times the Edge is part of Multi-Layer Butterfly Structure Type 5 |
| | 16 | MultiButType6 | N | Number of times the Edge is part of Multi-Layer Butterfly Structure Type 6 |
| | 17 | MultiButSrcNodeNumType1 | N | Number of times the Source node is part of Multi-Layer Butterfly Structure Type 1 |
| | 18 | MultiButSrcNodeNumType2 | N | Number of times the Source node is part of Multi-Layer Butterfly Structure Type 2 |
| | 19 | MultiButSrcNodeNumType3 | N | Number of times the Source node is part of Multi-Layer Butterfly Structure Type 3 |
| | 20 | MultiButSrcNodeNumType4 | N | Number of times the Source node is part of Multi-Layer Butterfly Structure Type 4 |
| | 21 | MultiButSrcNodeNumType5 | N | Number of times the Source node is part of Multi-Layer Butterfly Structure Type 5 |
| high-order | 22 | MultiButSrcNodeNumType6 | N | Number of times the Source node is part of Multi-Layer Butterfly Structure Type 6 |
| | 23 | MultiButDstNodeNumType1 | N | Number of times the Destination node is part of Multi-Layer Butterfly Structure Type 1 |
| | 24 | MultiButDstNodeNumType2 | N | Number of times the Destination node is part of Multi-Layer Butterfly Structure Type 2 |
| | 25 | MultiButDstNodeNumType3 | N | Number of times the Destination node is part of Multi-Layer Butterfly Structure Type 3 |
| | 26 | MultiButDstNodeNumType4 | N | Number of times the Destination node is part of Multi-Layer Butterfly Structure Type 4 |
| | 27 | MultiButDstNodeNumType5 | N | Number of times the Destination node is part of Multi-Layer Butterfly Structure Type 5 |
| | 28 | MultiButDstNodeNumType6 | N | Number of times the Destination node is part of Multi-Layer Butterfly Structure Type 6 |
| | 29 | BitrussLayer1 | N | Edge Bitruss Number in Layer 1 |
| | 30 | BitrussLayer2 | N | Edge Bitruss Number in Layer 2 |
| | 31 | SrcinNumberBloomsLayer1 | N | Number of Blooms source node is part of in Layer 1 |
| | 32 | SrcinNumberBloomsLayer2 | N | Number of Blooms source node is part of in Layer 2 |
| | 33 | DstinNumberBloomsLayer1 | N | Number of Blooms destination node is part of in Layer 1 |
| | 34 | DstinNumberBloomsLayer2 | N | Number of Blooms destination node is part of in Layer 2 |

Table 3.1: Multi-layer bipartite edge graph embedding features.

# Chapter 4

# Implementation

In this study, we measure the effectiveness of the simple first-order and the higher-order structural features (table 3.1), for the link prediction problem. We compare the structural features' predictive power over multi-layer bipartite graphs on YouTube.

To evaluate these features' predictive power, we use them in two settings. The first setting predicts the formation of future links. To predict a day's links on the YouTube network, we examine two cases. In case one, we gather one day's prior dense sample of the YouTube network, and in case two, a sample of two prior consecutive days. We then compare these two cases' predictions and find that to predict a day's interactions, it is better to take into account network information generated one day prior, i.e., most information can be found on the previous day. For the second setting, we predict missing links from the network. We use a YouTube network sample and hide 30% of its links. Then with the information of the rest of the network, we predict these hidden links. We found that the higher-order structural features can predict with 99% precision missing and future interactions in the comment network.

## 1 Dataset

We use the YouTube Data API to monitor YouTube traffic generated between 19 of July 2020 and 22 of September 2020 on videos related to the US 2020 elections. To gather the topic-specific videos, we first use the Twitter API to obtain tweets that contain hashtags and keywords related to the US elections. From the corpus of those tweets, we extract the YouTube video links and gathered a dense sample of the YouTube graph, instead of a sparse random sample of the whole graph. The dataset collection is explained in detail in [21].

The resulting dataset contains 12,538 videos. Those videos have 3,091,176 unique commenters and 27,927,909 comments and replies. From this YouTube dataset, we extract 2 bipartite graphs. The first graph represents the relation 'own', is a directed bipartite graph of *user owns video*, we define this graph to be the *Layer 1*. The second graph represents the relation 'comment', is a directed bipartite graph of *user comments video*, we define this graph to be the *Layer 2*. YouTube video and comment objects returned by the YouTube API are dated.

In this work, we explore both link prediction tasks, the prediction of the formation of future links, and the missing links predictions in a network. For both tasks, the goal is to predict if a user is to comment on a video. Specifically, we predict links on the YouTube 'comment' network, the Layer 2 in our setting.

## 2 Future Link Prediction

To predict the formation of future links, we randomly selected five different days of the gathered YouTube dataset between the three months of 19/7/20 and 22/9/20. To evaluate the predictive power of the proposed first and higher-order structural features (table 3.1), for each day of the five days, we construct 3 data sub-sets:

- we extract all data gathered for the day and call it the ***PredictSet***,
- we extract all data gathered for the previous day and call it the $\boldsymbol{TrainSet_1}$,
- we extract all data gathered for the two previous days and call it the $\boldsymbol{TrainSet_2}$.

These 3 data sub-sets include Layer 1 and Layer 2 relations, for the 'own' and 'comment' network, respectively.

We use $TrainSet_1$ to generate an embedding using the structural features shown in table 3.1. We then train a Logistic regression model on the embedding and evaluate the predictive power over the *PredictSet*. We repeat the same procedure for $TrainSet_2$.

Table 4.1 shows the structural characteristics of the five sets of data for the task of predicting links in the future. The first column (*No.*) refers to the dataset id. The second column (*Day (2020)*) is the date of each dataset. The third column (*Datset*) is the name of each data sub-set. The fourth and seventh columns ($V(G)L_1$, $V(G)L_2$) shows the number of nodes in Layer 1 and Layer 2, respectively in the set V. The fifth and eighth columns ($U(G)L_1$, $U(G)L_2$), shows the number of nodes in Layer 1 and Layer 2, respectively, in the set U. The sixth and ninth column ($EdgesL_1$, $EdgesL_2$) shows the number of edges in Layer 1 and Layer 2, respectively. The tenth column *Butterflies*$L_2$ shows the number of butterflies in Layer 2. The eleventh column *Blooms*$L_2$ shows the number of blooms in Layer 2. There are no records in the table for butterflies in Layer 1 because the relation 'own' cannot form this motif in the network, as one video cannot be owned by multiple users. Since there are no butterflies in Layer 1 there are no Bloom structures either.

We train two Logistic Regression models, one embedding containing only the first-order structural features and we call it the *'base model'*, and one with both first-order and high-order structural features, and call it the *'enhanced model'*. Following, we evaluate and compare the predictive power of the two models and found that the enhanced model performs better. Specifically, for the future link prediction task the base model's precision is 97%, and the enhanced model's precision is 99%.

| No. | Day (2020) | Dataset | V(G)$L_1$ | U(G)$L_1$ | Edges$L_1$ | V(G)$L_2$ | U(G)$L_2$ | Edges$L_2$ | Butterflies$L_2$ | Blooms$L_2$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 07/08 | PredictSet | 1,082 | 2,568 | 2,568 | 143,678 | 2,568 | 179,306 | 1,573,999 | 4,144 |
| | 06/08 | TrainSet$_1$ | 1,076 | 2,457 | 2,457 | 141,665 | 2,457 | 178,945 | 1,496,857 | 4,493 |
| | 05/08 & 06/08 | TrainSet$_2$ | 1,286 | 2,999 | 2,999 | 263,575 | 2,999 | 365,717 | 8,734,962 | 10,439 |
| 2 | 30/08 | PredictSet | 1,316 | 3,500 | 3,500 | 111,391 | 3,500 | 136,837 | 2,097,625 | 2,827 |
| | 29/08 | TrainSet$_1$ | 1,284 | 3,439 | 3,439 | 112,213 | 3,439 | 133,618 | 338,131 | 2,862 |
| | 28/08 & 29/08 | TrainSet$_2$ | 1,582 | 4,240 | 4,240 | 227,028 | 4,240 | 302,026 | 3,537,029 | 9,199 |
| 3 | 19/09 | PredictSet | 1,451 | 4,005 | 4,005 | 149,229 | 4,005 | 4,005 | 1,929,368 | 5,884 |
| | 18/09 | TrainSet$_1$ | 1,433 | 4,031 | 4,031 | 166,709 | 4,031 | 219,055 | 2725,572 | 6,932 |
| | 17/09 & 18/09 | TrainSet$_2$ | 1,785 | 5,136 | 5,136 | 286,263 | 5,136 | 416,425 | 10,365,662 | 16,070 |
| 4 | 04/09 | PredictSet | 866 | 501 | 866 | 10,059 | 866 | 10,296 | 35 | 15 |
| | 03/09 | TrainSet$_1$ | 454 | 795 | 795 | 12,655 | 795 | 12,916 | 247 | 24 |
| | 02/09 & 03/08 | TrainSet$_2$ | 4,315 | 1,523 | 4,315 | 269,916 | 4,315 | 403,965 | 8,107,893 | 13,325 |
| 5 | 24/07 | PredictSet | 899 | 1,796 | 1,796 | 95,570 | 1,796 | 112,827 | 661,254 | 1,803 |
| | 23/07 | TrainSet$_1$ | 866 | 1,725 | 1,725 | 80,839 | 1,725 | 92,214 | 383,554 | 1,188 |
| | 22/07 & 23/07 | TrainSet$_2$ | 2,141 | 1,080 | 2,141 | 156,443 | 2,141 | 189,557 | 1,637,450 | 3,477 |

Table 4.1: Structural characteristics of the 5 dataset used for the link prediction task.

# 3 Missing Link Prediction

For the missing link task, we select from the 5 random days described above, the data of TrainSet$_2$ (table 4.1). We split each of the 5 datasets into 70% *train-set* and 30% *predict-set*. For both sets we generate an embedding using the structural features shown in table 3.1 and train a Logistic regression model.

Following, we compare the predicting performance of the model trained only with an embedding containing the first-order structural features ('base model'), to the model ('enhanced model') trained with all the features from table 3.1. We found that the precision of 'base model' is 95%, and the precision of the 'enhanced model' is 99%. Having more information about the YouTube network with the 'enhanced model', is better to find missing link interactions on the YouTube network.

# Chapter 5

# Results

## 1 Future Link Prediction

For the future link prediction problem, we address the following two questions:

- *'How many prior days' (1 or 2) of data are needed to predict if a user is going to comment on a video tomorrow?'*
- *'What is the extra predictive value of the higher-order structural features over the first-order structural features?'*

Table 5.1 shows the results for the five experiments and their average (Avg) values for Precision, F1 score, and Execution time in seconds, which includes Train and Predict time. When we train with one day's data ($\text{TrainSets}_1$) for the base model, which only includes first-order structural features, we get 97% precision and the execution time takes 55 seconds. For $\text{TrainSets}_2$, which have two days' data on the base model we get an average precision of 97% and the execution time is 59 seconds. When we make predictions with the enhanced model, which includes first and high-order structural features, for the $\text{TrainSets}_1$ average precision is 99%, and the execution time is around 8 hours. For the $\text{TrainSets}_2$, we get average precision equal to 99%, and the execution time is around 18 hours.

We found that to make predictions for a day's interactions, suffice to account for network information generated 1 day prior. When making predictions with the enhanced model, which includes the higher-order structures, the average precision is 99% for the five experiments. It is more time-efficient to make predictions with 1 day's data. With 1 day's data, the prediction is on average 56% faster than with 2 days' data in the enhanced model. We also found that the higher-order structural features perform 2% better with 99%

average precision, compared to the first-order features with average precision 97%.

From the above, we infer for the YouTube network the following:

- for more precise future link predictions is better to use the enhanced model (all structural features) with precision 99%,
- the base model (only first-order structural features) can make predictions with 97% precision, and
- using one additional day's worth of data does not increase precision while adding 128% of running time when training with two days' data.

| | | Base model | | | Enhanced model | | |
|---|---|---|---|---|---|---|---|
| No. | Dataset | Precision | F1 score | Execution Time* | Precision | F1 score | Execution Time* |
| 1 | $\text{TrainSet}_1$ | 0.98 | 0.98 | 58 | 0.99 | 0.99 | 42,963 |
| | $\text{TrainSet}_2$ | 0.98 | 0.98 | 60 | 0.99 | 0.99 | 78,061 |
| 2 | $\text{TrainSet}_1$ | 0.97 | 0.97 | 62 | 0.99 | 0.99 | 30,019 |
| | $\text{TrainSet}_2$ | 0.97 | 0.97 | 66 | 0.99 | 0.98 | 60,496 |
| 3 | $\text{TrainSet}_1$ | 0.98 | 0.99 | 58 | 0.99 | 0.99 | 51,787 |
| | $\text{TrainSet}_2$ | 0.98 | 0.98 | 58 | 0.99 | 0.98 | 92,921 |
| 4 | $\text{TrainSet}_1$ | 0.98 | 0.98 | 45 | 0.98 | 0.98 | 844 |
| | $\text{TrainSet}_2$ | 0.98 | 0.98 | 55 | 0.99 | 0.98 | 65,991 |
| 5 | $\text{TrainSet}_1$ | 0.96 | 0.97 | 54 | 0.99 | 0.99 | 19,788 |
| | $\text{TrainSet}_2$ | 0.96 | 0.97 | 58 | 0.99 | 0.99 | 33,959 |
| | | Avg Precision | Avg F1 score | Avg Exec Time* | Avg Precision | Avg F1 score | Avg Exec Time* |
| | $\text{TrainSets}_1$ | 0.97 | 0.98 | 55 | 0.99 | 0.99 | 29,080 |
| | $\text{TrainSets}_2$ | 0.97 | 0.98 | 59 | 0.99 | 0.99 | 66,285 |

* Execution Time in seconds

Table 5.1: Future Link prediction results for the 5 dataset, when trained with base and enhanced model.

# 2 Missing Link Prediction

For the missing link prediction problem, we address the questions *'What is the extra predictive value of the higher-order structural features over the first-order structural features?'*

In this task, we select from the 5 random days described in section 2, the data of $\text{TrainSet}_2$ (table 4.1) and splitted each of the 5 datasets into 70% *train-set* and 30% *predict-set*. We selected the data of $\text{TrainSets}_2$ because it contains more information about the network to test the higher-order features to the first-order features.

Table 5.2 shows the results for the five experiments and their average (Avg) values for Precision, F1 score, and Execution time in seconds, which includes Train and Predict time. For this task, we only use the dataset $\text{TrainSet}_2$ that includes 2 days' network information.

In this predicting task, we again compare the base model, which only includes first-order structural features, to the enhanced model, which includes all of the structural features. When predicting with the base model we get an average precision of 95% and the running time is 35 seconds. With the enhanced model, the average precision is 99% and the execution time is around 9 hours.

We found for this task on the YouTube comment network on two days' interactions, is better to use the enhanced model, which includes the higher-order structures. The higher-order structural features perform 4% better than the first-order structural features with an average precision 99%.

From the above, we infer for the YouTube network the following:

- for more precise missing link predictions is better to use the enhanced model (all structural features) with precision 99%,
- using the enhanced model increases precision while adding a 99% increase of running time.

| | | Base model | | | Enhanced model | | |
|---|---|---|---|---|---|---|---|
| No. | Dataset | Precision | F1 score | Execution Time* | Precision | F1 score | Execution Time* |
| 1 | $\text{TrainSet}_2$ | 0.94 | 0.96 | 34 | 0.99 | 0.99 | 59,165 |
| 2 | $\text{TrainSet}_2$ | 0.96 | 0.97 | 38 | 0.99 | 0.99 | 41,357 |
| 3 | $\text{TrainSet}_2$ | 0.95 | 0.96 | 36 | 0.99 | 0.99 | 27,399 |
| 4 | $\text{TrainSet}_2$ | 0.95 | 0.96 | 33 | 0.98 | 0.99 | 24,682 |
| 5 | $\text{TrainSet}_2$ | 0.95 | 0.97 | 34 | 0.99 | 0.99 | 21,649 |
| | | Avg Precision | Average F1 score | Avg Exec Time* | Avg Precision | Avg F1 score | Avg Exec Time* |
| | $\text{TrainSets}_2$ | 0.95 | 0.97 | 35 | 0.99 | 0.99 | 34,850 |

* Execution Time in seconds

Table 5.2: Missing Link prediction results for the 5 $\text{TrainSet}_2$ dataset, when trained with base and enhanced model.

# 3 Set-up & Tools

For our experiments, we used a cluster of 5 servers with 32-core Intel(R) Xeon(R) E5-2630 CPUs and 256GB of main memory each, configured as 1 Spark Driver and 4 Spark Workers containing 3 Executors each. Each Executor used 83GB of memory and 10 cores, resulting in 120 total cores. Nodes connect with a 40Gb network.

# Chapter 6

# Conclusion

In this study, we measure how accurately we can predict comment interactions on the YouTube bipartite network, concerning predicting missing or unseen interactions. We propose high-order structural bipartite features, which enhance the performance of simple prediction models, to find missing or unseen links, which have never been used prior in the literature. Through experimentation on multi-layer networks, we find that we can predict future and unseen comment interactions on YouTube videos with precision 99%. We show that to predict a day's comment interactions it suffices to account network information generated 1 day prior. Finally, we implement 2 MapReduce algorithms, one that counts the bitruss number of an edge, and one that identifies blooms in bipartite networks.

# Bibliography

[1] Top 100: The most visited websites in the us [2021 top websites edition].

[2] Sinan G Aksoy, Tamara G Kolda, and Ali Pinar. Measuring and modeling bipartite graphs with community structure. *Journal of Complex Networks*, 5(4):581–603, 2017.

[3] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. Link prediction using supervised learning. In *SDM06: workshop on link analysis, counter-terrorism and security*, volume 30, pages 798–805, 2006.

[4] Mohammad Al Hasan and Mohammed J Zaki. A survey of link prediction in social networks. In *Social network data analytics*, pages 243–275. Springer, 2011.

[5] Oussama Allali, Clémence Magnien, and Matthieu Latapy. Link prediction in bipartite graphs using internal links and weighted projection. In *2011 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pages 936–941. IEEE, 2011.

[6] Nesserine Benchettara, Rushed Kanawati, and Celine Rouveirol. Supervised machine learning applied to link prediction in bipartite social networks. In *2010 international conference on advances in social networks analysis and mining*, pages 326–330. IEEE, 2010.

[7] Mustafa Bilgic, Galileo Mark Namata, and Lise Getoor. Combining collective classification and link prediction. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)*, pages 381–386. IEEE, 2007.

[8] Hsinchun Chen, Xin Li, and Zan Huang. Link prediction approach to collaborative filtering. In *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'05)*, pages 141–142. IEEE, 2005.

[9] Xu Cheng, Cameron Dale, and Jiangchuan Liu. Statistics and social network of youtube videos. In *2008 16th Interntional Workshop on Quality of Service*, pages 229–238, 2008.

[10] Janardhan Rao Doppa, Jun Yu, Prasad Tadepalli, and Lise Getoor. Chance-constrained programs for link prediction. In *NIPS workshop on analyzing networks and learning with graphs*. Citeseer, 2009.

[11] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 07 2014.

[12] Maksim Koptelov, Albrecht Zimmermann, Bruno Crémilleux, and Lina Soualmia. Link prediction via community detection in bipartite multilayer graphs. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 430–439, 2020.

[13] Jérôme Kunegis, Ernesto W De Luca, and Sahin Albayrak. The link prediction problem in bipartite networks. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems*, pages 380–389. Springer, 2010.

[14] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–470, 2008.

[15] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World wide web*, pages 641–650, 2010.

[16] Xin Li and Hsinchun Chen. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems*, 54(2):880–890, 2013.

[17] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[18] Huda Nassar, Austin R Benson, and David F Gleich. Pairwise link prediction. In *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 386–393. IEEE, 2019.

[19] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. Butterfly counting in bipartite networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2150–2159, 2018.

[20] Ahmet Erdem Sarıyüce and Ali Pinar. Peeling bipartite networks for dense subgraph discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 504–512, 2018.

[21] Alexander Shevtsov, Maria Oikonomidou, Despoina Antonakaki, Polyvios Pratikakis, and Sotiris Ioannidis. Analysis of twitter and youtube during uselections 2020. *arXiv preprint arXiv:2010.08183*, 2020.

[22] Konstantinos Sotiropoulos, John W Byers, Polyvios Pratikakis, and Charalampos E Tsourakakis. Twittermancer: predicting user interactions on twitter. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 973–980. IEEE, 2019.

[23] Chao Wang, Venu Satuluri, and Srinivasan Parthasarathy. Local probabilistic models for link prediction. In *Seventh IEEE international conference on data mining (ICDM 2007)*, pages 322–331. IEEE, 2007.

[24] Jia Wang, Ada Wai-Chee Fu, and James Cheng. Rectangle counting in large bipartite graphs. In *2014 IEEE International Congress on Big Data*, pages 17–24. IEEE, 2014.

[25] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. Vertex priority based butterfly counting for large-scale bipartite networks. *PVLDB*, 2019.

[26] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. Efficient bitruss decomposition for large-scale bipartite graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 661–672. IEEE, 2020.

[27] Mirjam Wattenhofer, Roger Wattenhofer, and Zack Zhu. The youtube social network. In *Sixth international AAAI conference on weblogs and social media*, 2012.

[28] Hema Yoganarasimhan. Impact of social network structure on content propagation: A study using youtube data. *Quantitative Marketing and Economics*, 10(1):111–150, 2012.