

# Vertex Similarity in Graphs using Feature Learning

Group mi109f17  
Aalborg University  
Department of computer science





**AALBORG UNIVERSITY**  
STUDENT REPORT

**Cassiopeia**  
**House of Computer Science**  
Selma Lagerlöfs Vej 300  
Telephone 99 40 99 40  
Fax 99 40 97 98  
<http://www.cs.aau.dk>

**Title:**

Vertex Similarity in Graphs using  
Feature Learning

**Project period:**

DAT10, spring semester 2017

**Project group:**

mi109f17

**Members:**

Anders Hermansen  
Bjarke Thorn Carstens  
Mads Riis Jensen  
Mathias Friis Spaniel

**Supervisor:**

Manfred Jaeger

**Total pages:** 89 pages

**Appendix pages:** 1 pages

**Ended at:** June 7, 2017

**Synopsis:**

In this report we investigate node2vec, a feature learning model for mapping vertices to euclidean space. node2vec utilizes only the identifiers of vertices and thus does not take the attributes available in real networks into account. We show how node2vec can be extended to utilize network information of both vertices and edges. The authors of node2vec claim parameters  $p$  and  $q$  can be tuned, to decide what kind of similarity is found. In our detailed investigation of node2vec, we find that  $p$  and  $q$  barely affects what is found, and that the length of walks and number of walks influence what type of similarity is found more. We evaluate our extensions against node2vec, and other methods, using macro and micro F1-score on real networks on multi class and multi label classification, and find that at least one of our extensions always perform better than node2vec in terms of F1-score.

*The content of the report is freely available, but publication (with source-reference) is only possible with an agreement from the authors.*



---

---

# Preface

---

---

This article is a tenth semester master thesis, written by a group of four students at Aalborg University at the Computer Science department. The article and experiments were performed in the period 2. February 2017 to the 7. June 2017.

---

Anders Hermansen

---

Bjarke Thorn Carstens

---

Mads Riis Jensen

---

Mathias Friis Spaniel

June 7, 2017



## Summary

Many real-life networks, such as social networks, can be represented as graphs. A task one could consider doing on these graphs, is to measure the similarity between its vertices. There exists different types of similarity that we can measure, such as structural and semantic vertex similarity. Structural similarity has to do with the structure around vertices and identifying roles. One role which exists in many networks are hubs. For semantic similarity, we are concerned with the attributes of vertices and edges, and say that two vertices are semantically similar if they have the same attributes. Often it is not enough to base the similar on one of these concepts, but instead a combination of them, i.e. using both structure and attributes. Being able to measure this kind of similarity has numerous applications. For example classifying proteins in a protein-protein interaction network.

Recently, methods for measuring similarity have been proposed, which use feature learning. Feature learning techniques learn embeddings for vertices in graphs. These techniques often do this by creating random walks and use these as input to the Skip-gram framework.

We explore in detail one such method, namely, node2vec. node2vec is a framework for learning embeddings for vertices in graphs, i.e. mapping vertices to euclidean space. To do this, they formalize an objective function closely related to the objective function of Skip-gram, which aims to predict the neighbourhood of a given vertex. They use biased random walks to sample the neighbourhood.

The authors claim their biased random walks, configurable by two parameters,  $p$  and  $q$ , are able to determine if the sampled neighbourhoods corresponds to structure or proximity.

One drawback of node2vec is that it ignores vertex attributes, and uses only the ids of vertices. We propose ways of extending node2vec to utilize attributes on vertices and edge relations given a multirelational graph. A walk in node2vec is represented as a sequence of vertex ids, to utilize attributes, the walk should instead be represented as a sequence of feature vectors.

In our exploratory analysis we examine the claims of node2vec, namely, that they are capable of finding structural similarity or proximity by configuring  $p$  and  $q$ . To examine this claim, we find a set of parameters, which are able to find structural similarity, and see how changing the parameters affect its ability to find structure. Specifically, we see how well it distinguishes between three types of network roles, namely, hubs, periphery and

mainstream vertices. This is done using k-means, where the ideal clustering is three clusters with each cluster containing only one type of the previously mentioned network role. We found that node2vec were able to correctly distinguish between the different roles, however,  $p$  and  $q$  did not decide whether structure or proximity were found. Instead the length and number of biased random walks, were found to have huge influence in whether structure or proximity were found. We also performed experiments to investigate the effect of adding attribute and edge relations to the random walks. We found that it made it possible to discover semantic similarity.

Having explored node2vec in detail, we set out to perform experiments on real networks, to test how our extension performed in contrast to node2vec on real tasks. These tasks include multiclass and multilabel classification. We found that for the multiclass task that one of our extensions performed better than node2vec. For the multilabel classification problem results were mixed. On some datasets node2vec performed best while on another our extensions performed better.



# Contents

## Contents

<b>I</b>	<b>Introduction</b>	<b>1</b>
<b>1</b>	<b>Introduction and motivation</b>	<b>1</b>
1.1	Use cases . . . . .	3
1.2	Outline . . . . .	4
<b>2</b>	<b>Graph Theory</b>	<b>4</b>
2.1	Multirelational graphs . . . . .	5
2.2	Graph properties . . . . .	5
2.3	Graph statistics . . . . .	6
<b>II</b>	<b>Similarity</b>	<b>9</b>
<b>3</b>	<b>Similarity</b>	<b>9</b>
3.1	Types of Similarity . . . . .	10
3.2	Structural Vertex similarity . . . . .	11
3.3	Structural and Semantic similarity . . . . .	13
3.4	Structural similarity in multirelational graphs . . . . .	14
<b>4</b>	<b>Methods</b>	<b>17</b>
4.1	Isomorphism . . . . .	17
4.2	Graph edit distance . . . . .	19
4.3	Graph Kernels . . . . .	19
4.4	Feature based methods . . . . .	23
4.5	Feature learning . . . . .	26
<b>5</b>	<b>Similarity measure criteria</b>	<b>33</b>
<b>III</b>	<b>Exploratory analysis</b>	<b>37</b>
<b>6</b>	<b>Procedure</b>	<b>37</b>
6.1	K-means . . . . .	37
6.2	Adjusted rand index . . . . .	38
6.3	Graph cluster cohesion . . . . .	38
<b>7</b>	<b>Dataset overview</b>	<b>38</b>
7.1	Airline . . . . .	39
7.2	Zhachary Karate Club . . . . .	41
7.3	Les Misérables . . . . .	41

<b>8</b>	<b>Node2vec unattributed graph experiments</b>	<b>41</b>
8.1	Structural parameters . . . . .	41
8.2	Changing parameters . . . . .	46
8.3	Les Misérables . . . . .	55
<b>9</b>	<b>Node2vec attributed experiments</b>	<b>58</b>
9.1	5Comm . . . . .	58
9.2	Experiments . . . . .	59
<b>10</b>	<b>Case study: Airline network</b>	<b>64</b>
10.1	Airline embeddings . . . . .	64
10.2	Airport search . . . . .	68
<b>IV</b>	<b>Experimental study</b>	<b>75</b>
<b>11</b>	<b>Experimental procedure</b>	<b>75</b>
11.1	Vertex classification . . . . .	75
11.2	Evaluation measures . . . . .	75
11.3	Methods used . . . . .	76
<b>12</b>	<b>Dataset overview</b>	<b>77</b>
12.1	BlogCatalog . . . . .	77
12.2	Wikipedia . . . . .	77
12.3	PPI . . . . .	77
<b>13</b>	<b>Airline results</b>	<b>78</b>
<b>14</b>	<b>Multilabel classification</b>	<b>81</b>
<b>15</b>	<b>Scalability</b>	<b>85</b>
<b>V</b>	<b>Conclusion</b>	<b>87</b>
	<b>Bibliography</b>	<b>88</b>
	<b>Appendix A</b>	<b>90</b>



# Vertex Similarity in Graphs using Feature Learning

Bjarke Thorn Carstens    Anders Hermansen    Mads Riis Jensen  
Mathias Friis Spaniel

June 7, 2017

## Part I

# Introduction

## 1 Introduction and motivation

Real life networks, such as social networks, can be represented as graphs. In these graphs the entities of the network, e.g. users of a social network, are vertices and the relations between these entities, e.g. friendship, are edges.

In these networks it is sometimes of interest to measure how similar two vertices are. Some tasks where measuring vertex similarity is useful involve identifying vertices with a certain role, e.g. teacher and students, searching for certain vertices, or classifying vertices according to some label.

There exists many different notions on what makes two vertices similar. For example, two users in a social network could be considered similar if they have many of the same friends. We are particularly interested in a type of similarity called structural similarity. This type of similarity is based on the structural patterns of a vertex.

Structural similarity is useful for determining the roles of vertices. As an example, Figure 1 shows the Zachary Karate network, where vertices are karate instructors and students, and edges are social interactions between them. The two highlighted vertices, 0 and 32, are both instructors. They are considered structurally similar because they both have a high degree and connect to many low degree vertices. This makes sense, because instructors have interactions with many different students, while students mainly interact with the instructor and a few other students.

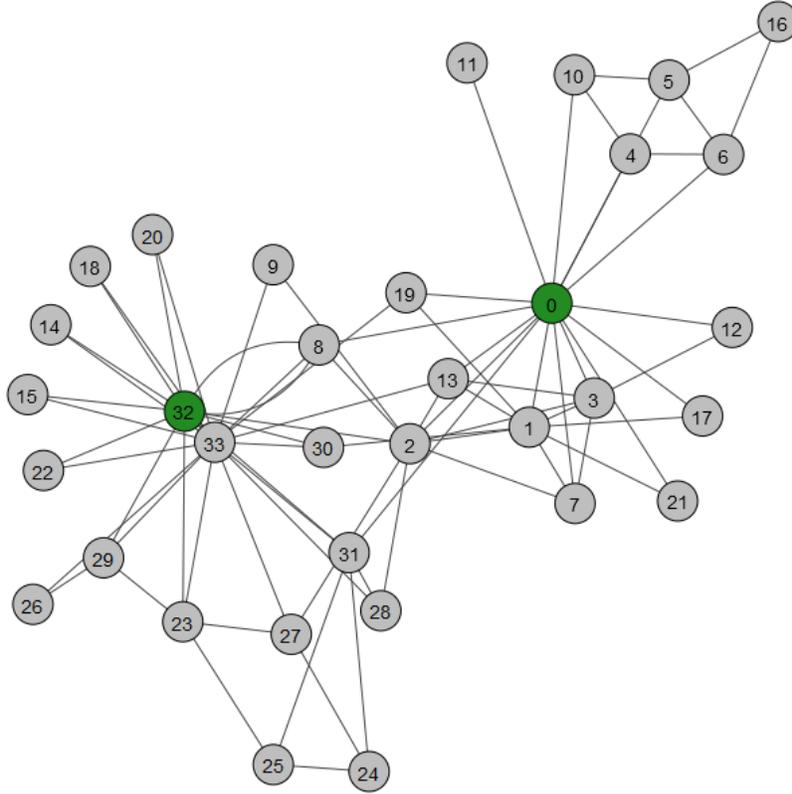


Figure 1: Example graph with two structurally similar vertices highlighted

Vertices in real networks generally have several attributes, e.g. a user in a social network may have a name and an age. In the karate network, an attribute could be which karate club a student or instructor is a member of. We refer to similarity based on attributes, as semantic similarity.

Combining both of these types of similarity is interesting, because it makes it possible to distinguish between more detailed roles. With structural similarity we are only able to distinguish instructors from students. By combining it with semantic similarity we are able to distinguish between instructors of one club and instructors of another club.

Some real networks have multiple relation types. For example, a social network may have a friend relation and a family relation. Like attributes, relation types can also give relevant information about the role of a vertex. For example, consider a network where vertices are airports and edges are flight routes between airports. A relation type would be the airline that operates the route. In this network, taking the relation type into consideration makes it possible to distinguish between hub airports from different airlines.

State of the art techniques for measuring similarity between vertices make use of feature learning. The goal of feature learning is to learn embeddings of vertices such that two vertices are similar if the distance between their embeddings is small. One method that makes use of feature learning is node2vec[6], which uses the structure of the graph to learn vertex embeddings. Since we wish to combine structural and

semantic similarity we extend `node2vec` to take attributes and relation types into account.

Our contributions are as follows:

- We perform an extensive analysis of `node2vec` and investigate the authors claims that `node2vec` can find both structural similarity and communities by changing two parameters.
- We propose extensions to `node2vec`, which makes it possible to utilize vertex attributes and edge relation types.
- We demonstrate that our extensions results in better classification performance than `node2vec` in some cases.

## 1.1 Use cases

In this section we highlight real-life use cases where vertex similarity measures may be useful.

### 1.1.1 Airline network

An airline network consists of airports and routes for different airlines. This can be represented as an attributed undirected multirelational graph, where airports are vertices and routes are edges. Each relation is associated with one airline. Attributes include passengers carried, and identification code, e.g. International Air Transport Association. Vertex similarity can be used for performing search in this network. If we are given an airport of a certain size, the search problem might entail finding other airports of similar size.

### 1.1.2 Social network

There are many opportunities for vertex similarity to be useful, when considering real-life social network data, such as a blogging network or Facebook.

For example, consider a blogging network. This can be represented as an attributed undirected graph, where vertices are blogs and edges exist between two blogs, if the authors of the blogs are friends. In such a network, attributes may include when the blogs was posted, tags or keywords describing the blog, and the text of the blog. An interesting task to perform on this network, could be classification of tags. Vertex similarity could be utilized here, as blogs that are similar in terms of their attributes and their structure, should have similar tags. If a blog is posted without any tags, this could be used for automatic tagging.

Additionally, it might be possible to use this for friend recommendation, as users who share similar interest(i.e. writing blogs with similar tags), might be candidates for being friends.

## 1.2 Outline

The report is structured as follows. In section 2 we present the concepts used from graph theory. In section 3 we explore similarity in detail. In section 4 we highlight ways of measuring similarity. In section 5 we present different criteria one should think when choosing a similarity measure for their problem domain. In section 8, section 9, and section 10 we perform a comprehensive analysis of node2vec and our proposed extensions. In section 13 and section 14 we show the results of our experimental study.

## 2 Graph Theory

*This section, except 2.1 and 2.3.3, is largely based on last semesters report: Vertex Similarity in Graphs*

A graph is an ordered pair  $G = (V, E)$ , where  $V = \{v_1, v_2, \dots, v_n\}$  is the set of vertices in  $G$  and  $E \subseteq V \times V$  is the set of edges in  $G$ . We use  $V(G)$  to denote the vertices of graph  $G$  and  $E(G)$  to denote the edges of graph  $G$ . A vertex  $v_i$  is said to be adjacent, or neighbour, to a vertex  $v_j$  if  $v_i$  and  $v_j$  are connected by an edge, sometimes denoted  $v_i \sim v_j$ . An edge  $e_i$  is said to be incident on a vertex  $v_i$  if  $v_i$  is an endpoint of  $e_i$ . The size of  $G$ , denoted  $|G|$  is the amount of vertices in  $G$ .

A graph  $G$  can be represented as the adjacency matrix:

$$A_{ij} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E(G) \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

In a directed graph the set of edges consists of an ordered pair of vertices, while in an undirected graph the set of edges consists of unordered pairs of vertices. A *simple graph* is a graph without self loops and multiple edges.

A graph can have attributes on both vertices and edges. An attribute has a name and a value. Let  $\alpha$  be an attribute name and  $\Phi_\alpha$  be the range of values this attribute can take. A graph  $G$  then has a function  $\alpha : V(G) \rightarrow \Phi_\alpha$  for every attribute. The attribute functions for edges are defined similarly. An example of a numeric attribute could be *age* where  $\Phi_{age} = \mathbb{N}$ , and a discrete attribute could be *gender* where  $\Phi_{gender} = \{Male, Female\}$ . Graphs with a single numeric attribute on the edges are called weighted graphs.

A *walk*  $h$  on a graph  $G$  is a sequence of vertices and edges  $v_{h1}, e_{h1}, v_{h2}, \dots, e_{hn-1}, v_{hn}$  such that  $\forall i : (v_{hi} \in V(G) \wedge e_{hi} \in E(G) \wedge e_{hi} = (v_{hi}, v_{hi+1}))$ . The length of walk  $h$ , denoted  $|h|$ , is the number of edges in  $h$ . A *path* is a walk that does not contain any duplicate vertices i.e.  $\forall i, j : v_{hi} \neq v_{hj} \text{ iff } j \neq i$ . Two vertices are said to be connected if there exists a path between them.

A *subgraph*  $H$  of  $G$  is a graph where  $V(H) \subseteq V(G)$  and  $E(H) \subseteq E(G)$ . Subgraphs can be induced from vertices. A vertex-induced subgraph  $H'$  of  $G$  is a subgraph formed by a subset of vertices  $V(H') \subseteq V(G)$  and  $E(H') = \{(v_i, v_j) \in E(G) \mid v_i \in V(H') \wedge v_j \in V(H')\}$ .

The  $k$ -neighbourhood of a vertex  $v$ , denoted  $N_k(v)$ , is the subgraph induced by the set of all vertices that are reachable from  $v$  with a path of length at most  $k$ .

## 2.1 Multirelational graphs

In real networks two vertices can be connected by several different types of edges. For example, consider a social network where there are edges between people if they are friends or co-workers. These kind of networks are often called multirelational networks. Other closely related names for these kinds of networks are multidimensional, multilayer, and multiplex networks[3][8]. Formally a multirelational network is a triple  $G = (V, E, R)$  where  $V$  is a set of vertices,  $R$  is a set of relations, and  $E$  is a set of labelled edges,  $(u, v, r)$ , where  $u, v \in V$  and  $r \in R$ .

This is equivalent to a single-relational graph with an attribute *relation* :  $E \rightarrow R$  that maps edges to relations.

An example of a multirelational network can be seen in Figure 2. This network has three different relations,  $r_1$ ,  $r_2$  and  $r_3$ .

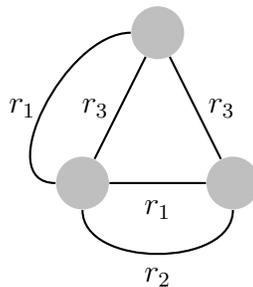


Figure 2: A multirelational network

## 2.2 Graph properties

Graphs have certain properties that are helpful when analysing the graph. This sections covers some of them.

### 2.2.1 Degree

The *degree* of a vertex  $v$  in an undirected graph, denoted  $deg(v)$ , is the number of edges that are incident on  $v$ . For a directed graph, we differentiate between *indegree*, denoted  $deg^-(v)$ , *outdegree*, denoted  $deg^+(v)$ , and degree  $deg(v) = deg^-(v) + deg^+(v)$ . The *indegree* is the number of incoming edges on  $v$ , and the *outdegree* is number of outgoing edges of  $v$ . In Figure 3, examples are illustrated of both an undirected and a directed graph, with each vertex labelled with its degree. Here the labels of the undirected graph specifies total degree for the given vertex, while vertex degrees for the directed graph are labelled as  $(deg^-(v), deg^+(v))$ .

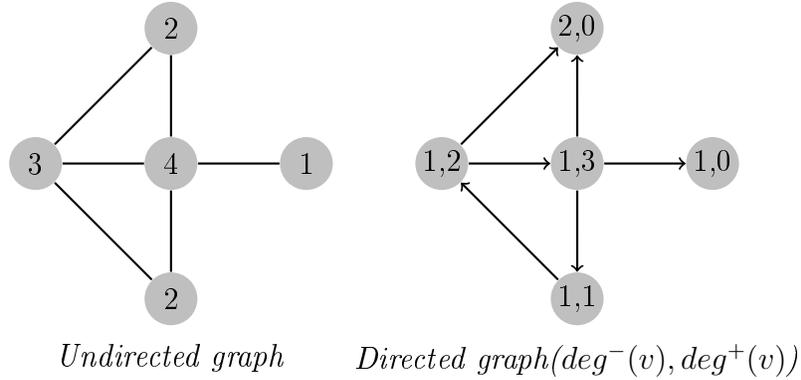


Figure 3: A directed and undirected graph with vertices labelled by their degrees

### 2.2.2 Distance

The distance between two vertices, denoted  $dist(v_i, v_j)$ , is defined as the number of edges in a shortest path from  $v_i$  to  $v_j$ . The shortest path is defined as the path from  $v_i$  to  $v_j$  with the least edges, or in case of a weighted graph, the path from  $v_i$  to  $v_j$  with the smallest accumulated weight. If no shortest path exists,  $dist(v_i, v_j)$  is defined to be infinite.

The *eccentricity* of a vertex  $v$ , denoted  $\epsilon(v) = \max_{v_i} dist(v, v_i)$ , is the largest shortest path distance between  $v$  and any other vertex. The *diameter* of a graph,  $dia(G) = \max_v \epsilon(v)$ , is the maximum  $\epsilon(v)$  for any  $v \in V(G)$ . Figure 4 shows an example of a graph  $G$  where the vertices are labelled with their *eccentricity*. This graph has a *diameter* of four, due to the largest  $\epsilon(v)$  in  $G$  being four.

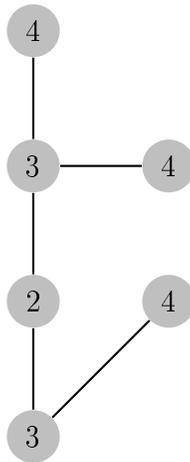


Figure 4: Graph  $G$  whose vertices are labelled with their eccentricity

## 2.3 Graph statistics

In this section we introduce common statistics used for graph analysis.

### 2.3.1 Closeness Centrality

The *closeness centrality* of a vertex  $v$  is a measure of how central  $v$  is located in a graph. It is determined by summing the lengths of the shortest paths between  $v$  and all other vertices in the graph. The closer  $v$  is to all other vertices, the larger its closeness centrality is. Formally, closeness centrality is defined in Equation 2.

$$C(v) = \frac{N - 1}{\sum_i^N dist_{sp}(v, v_i)} \quad (2)$$

Where  $C(v)$  is the closeness centrality of vertex  $v \in V(G)$ ,  $N$  is  $|G|$  and  $dist_{sp}(v, v_i)$  is the shortest path distance between  $v$  and  $v_i$ . Figure 5 shows an example of a graph, where vertices are labelled with their closeness centrality.

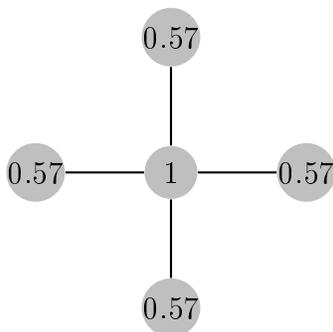


Figure 5: Graph  $G$  whose vertices are labelled with their closeness centrality

### 2.3.2 Graph Density

The density of a simple graph is defined by how close the graph is to having the maximal number of edges it can possibly have. Graph density is formally defined in Equation 3 and Equation 4 for undirected and directed graphs respectively.

$$D = \frac{2|E(G)|}{|V(G)|(|V(G)| - 1)} \quad (3)$$

$$D = \frac{|E(G)|}{|V(G)|(|V(G)| - 1)} \quad (4)$$

Where  $D$  is the density of a graph  $G$ .

### 2.3.3 Local clustering coefficient

The local clustering coefficient is used to quantify how connected the neighbours of a vertex is. Given a graph  $G = (V, E)$ , let  $k_v$  be the degree of  $v \in V$  and  $N_v$  denote the amount of links between the neighbours of  $v$ . Then the clustering coefficient is the ratio of connections in  $N_v$  over all possible connections that could exist between them. This is formally defined in Equation 5 for directed graphs, where  $k_v = deg^+(v)$ .

$$CC(v) = \begin{cases} \frac{N_v}{k_v(k_v-1)}, & \text{if } k_v > 1 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Here  $CC(v)$  is the clustering coefficient for vertex  $v$ . We say that the clustering coefficient of a vertex is zero, if it has less than two neighbours.

For undirected graphs the definition is slightly different, as a fully connected undirected graph can only contain half as many edges as a fully connected directed graph. Thus, for undirected graphs  $k_v(k_v - 1)$  would need to be divided by two.

In Figure 6 is an example of a directed and undirected graph respectively, whose vertices are labelled with their clustering coefficient.

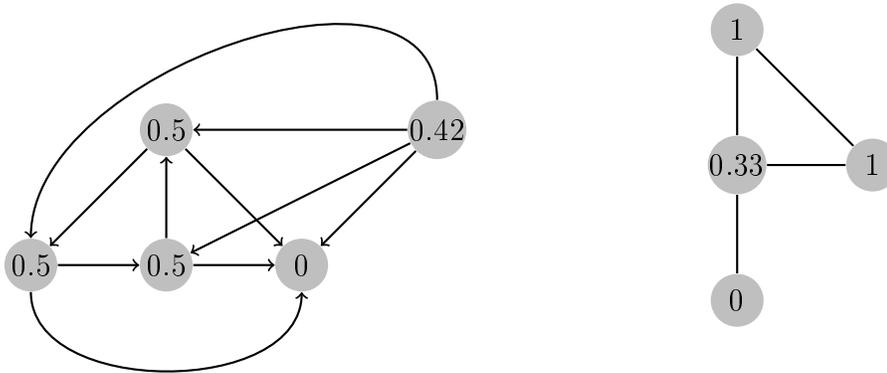


Figure 6: (Left) A directed graph with vertices labelled with their clustering coefficient, (Right) An undirected graph with vertices labelled with their clustering coefficient

## Part II

# Similarity

In this part we formally define what is understood by similarity, and what types of similarity exist. After that we show some of the popular methods for computing similarity on graphs and vertices. Next we introduce `node2vec`, a state of the art method which uses feature learning to learn vertex embeddings.

`Node2vec` does not make use of vertex attributes or relation types. We propose extensions to `node2vec` that makes use of these.

Finally we define some criteria that can be used to examine similarity measures. These criteria cover what types of data the similarity measures are applicable on, and how the measures behave on the data.

### 3 Similarity

A similarity measure is a function  $s(x, x')$  that gives a numerical quantifier on the similarity between the two objects,  $x$  and  $x'$ . The more similar two objects are, the larger  $s(x, x')$  is.

Most similarity measures exhibit certain properties. For example, similarity measures usually lie in the range  $[0, 1]$  or in some cases  $[-1, 1]$ . To ensure that values fall in these ranges, the measures may need to be normalized. A reason why it is attractive to have the values in these ranges is because it allows for comparison of results over different similarity measures.

Another property for these measures is that an object should be maximally similar to itself. More formally, for a similarity measure  $s$  we have that  $s(x, x') = 1$  if  $x = x'$  ( $0 \leq s \leq 1$ ).

A third property is that similarity measures should be symmetric  $s(x, x') = s(x', x)$ .

Similarity measures can often be transformed into a distance measure  $d(x, x')$ , and vice versa. In other words, a distance measure can be seen as the inverse of a similarity measure. For example, the euclidean distance between two points (defined in Equation 6) can be converted to a similarity measure by simply multiplying by -1.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n |x_i - y_i|^2} \quad (6)$$

Other ways to transform a similarity measure can be seen below, where  $s$  is a similarity measure and  $d$  is a distance measure.

$$s = 1 - d$$
$$s = \frac{1}{1 + d}$$

For network data we have measures that work on three different types of data: Graphs, edges and vertices. A graph similarity measure  $s_g(G_1, G_2)$  compares entire graphs, an edge similarity measure  $s_e((v_1, v_2), (v_3, v_4))$  compares two pairs of vertices, and a vertex similarity measure  $s_v(v_1, v_2)$  compares two vertices. In this report we are only interested in similarity measures between vertices.

In addition to different types of similarity measures, we also differentiate between different types of similarity. These will be discussed in the following section.

### 3.1 Types of Similarity

In real life networks there exist different types of similarity between vertices. We consider three different types, namely, *structural similarity*, *semantic similarity* and *proximity*.

From a structural perspective, the similarity depends only on the structure of the network. Examples of such vertex structures are hubs and outliers. As an example, two people in a social network would be considered structurally similar if they are both hubs.

Similarity in the semantic setting depends only on the attributes of vertices and edges. Again, consider a social network with people, and the only attribute we have available is the age of each person. Two people would be considered maximally semantically similar if they are of the same age, independently of the amount of friends they may have.

Finally, from a perspective of proximity, we have that vertices that are closely connected are more similar. So two people in a network would be considered similar from a proximity perspective if they are closely associated.

While these concepts define some form of basis for similarity, one might be interested in a measure that combines the different settings. Structurally dissimilar people may still receive a high similarity score because they have similar attributes.

To illustrate the different concepts, consider Figure 7. The two labels are attributes.

Vertices  $v_1$  and  $v_3$  are maximally semantically similar as they have the same attributes. However, they are only somewhat structurally similar as their amount of neighbours differ. Vertices  $v_1$  and  $v_7$  can be seen as somewhat similar from a semantic perspective as they share some attributes. However, their structure looks nothing alike. Vertex  $v_1$  is maximally structurally similar to  $v_6$ .

Based on proximity,  $v_1$  and  $v_5$  will be seen as very similar, even though both their structure and attributes are nothing alike. Utilizing the combination of these concepts might yield  $v_2$  as being the most similar vertex to  $v_1$ , as they not only share some attributes, they are also close to having a similar structure and are closely connected.

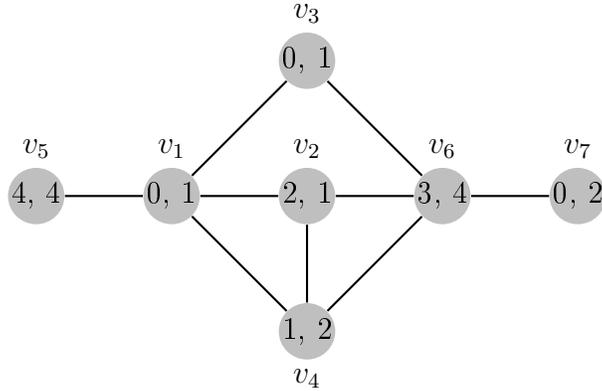


Figure 7: Example graph to illustrate different types of similarity

### 3.2 Structural Vertex similarity

There are several different notions on how to quantify the structural similarity between vertices. One commonly used notion is that two vertices are similar if they have many of the same neighbours [10]. For this notion, several well established similarity measures can be used. Let  $v$  and  $u$  and be two vertices in the same graph, then the following measures can be used:

$$s_{jaccard}(v, u) = \frac{|N_1(v) \cap N_1(u)|}{|N_1(v) \cup N_1(u)|} \quad (7)$$

$$s_{cosine}(v, u) = \frac{|N_1(v) \cap N_1(u)|}{\sqrt{|N_1(v)||N_1(u)|}} \quad (8)$$

This kind of similarity is based on what they in sociology refer to as *structural equivalence*, where two vertices are similar if they can be exchanged without changing any property of the graph. We refer to it as *proximity*. The drawback with this notion of similarity, is that for two vertices to be similar, they have to be close in the graph. In fact, vertices that are a distance of three or farther away, or disconnected from each other, will always have a similarity of zero, as they will never have any common neighbours.

Depending on the task, this kind of similarity may not be sufficient. There are several cases where two vertices should be considered structurally similar, but they do not have any neighbours in common. Consider the graph in Figure 8. Here the three vertices  $v_1$ ,  $v_2$ , and  $v_3$  are far away from each other, in fact, they are disconnected from each other. Yet, they all share similar structural properties in that they are the centre of a star.

A naive approach to construct a structural similarity measure that does not require connectivity, could be to utilize vertex properties such as closeness centrality and degree. For example, if we return to Figure 8, and consider the distance measure  $s_{degree}(v, u) = \sqrt{(deg(v) - deg(u))^2}$ . It is clear that the centres of the stars are closer to one and another, than to the rest of the vertices. In fact, we can see that  $v_1$  and

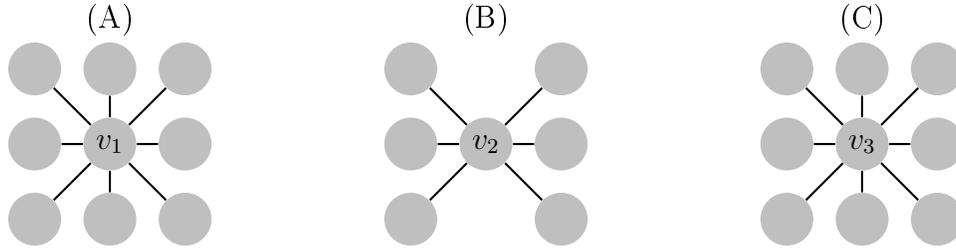


Figure 8: Example of a graph modelling three different learning institutions. Vertices are students and teachers. Edges exist between vertices if one is being taught by the other.

$v_3$  are identical using  $s_{degree}$ , as they have a distance of zero i.e.  $\sqrt{(8-8)^2} = 0$ . This measure allows us to differentiate between teachers and students, as teachers will be closer to other teachers than to students.

One concept of structural similarity is the idea of identifying roles in networks. Roles are specific structural patterns that often occur in real life networks. One notable such role, is a hub. A hub is a vertex which is highly connected in the network and has a significantly larger degree relative to its neighbours, as defined in Equation 9.

$$deg(v) \gg \frac{\sum_{v_n \in N_1(v)} deg(v_n)}{|N_1(v)|} \quad (9)$$

Here  $N_1(v)$  is the immediate neighbourhood of vertex  $v$  and  $v_n$  is a vertex in the immediate neighbourhood of  $v$ . For hubs it is sufficient to consider only the immediate neighbourhood, but for other structural patterns it may be beneficial to increase the neighbourhood size.

This definition highlights an important aspect of structural similarity, which is that it is not only dependent on the structural properties of an individual vertex, but also on the structure of its neighbouring vertices. However, note that determining roles and measuring similarity is not the same exact task. Focusing on Figure 8 again, it is clear that  $v_1$ ,  $v_2$ , and  $v_3$  are all hubs, as they have a much larger degree than their neighbours. However,  $v_1$  and  $v_3$  are clearly more structurally similar, than  $v_1$  and  $v_2$  as we saw before.

Closely related to the concept of hubs are two other roles, namely periphery and mainstream vertices. The definitions of these are more loose compared to hubs. For a vertex  $v$  to be considered a periphery vertex, it must have a distance of  $dia(G)$  for some graph  $G$  to some other vertex, i.e.  $\epsilon(v) = dia(G)$ . Periphery vertices generally have low degree, and are mostly connected to either hubs or mainstream vertices. Mainstream vertices are vertices which are neither hubs nor periphery vertices.

In the following section we will see how to combine structural and semantic similarity.

### 3.3 Structural and Semantic similarity

Utilizing only the structure of a network as a similarity measure may not be enough for some real networks. These networks often contain rich data such as attributes on vertices and edges, which are not utilized when looking purely at structure. For example, if we recall the graph modelling different institutions from the previous section, we could have that the teachers and students were annotated by their age, as seen in Figure 9.

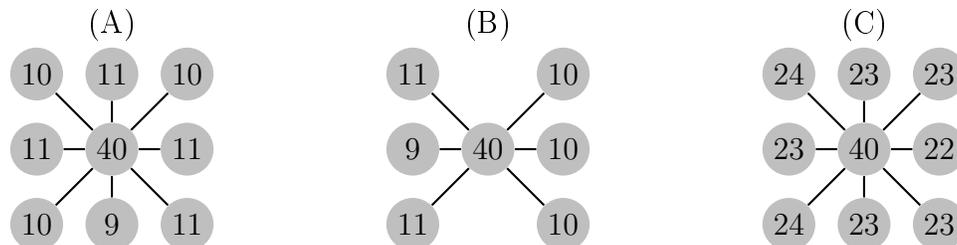


Figure 9: Example of a graph modelling three different institutions, (A) and (B) are elementary schools, and (C) is a university. Vertices are students and teachers and are labelled with their age. Edges exist between vertices if one is being taught by the other.  $v_1$  is the centre of (A),  $v_2$  is the centre of (B), and  $v_3$  is the centre of (C)

These attributes should be incorporated into a vertex similarity measure. The straightforward approach to this is to use one of the many similarity/distance measures for numeric values, such as Euclidean distance, to measure similarity of attributes. This measure can be combined with a structural similarity measure by either multiplying or adding the similarity scores together.

It is important to also consider the attributes of the neighbours when comparing vertices. If we do not consider the neighbours, we can see from Figure 9 that  $v_1$ ,  $v_2$  and  $v_3$  are all maximally similar to each other. This result is not any better than what we obtained from using just the structure, as we are still only able to differentiate between teachers and students.

If we use the attributes of the neighbours we can see that  $v_1$  and  $v_2$  are connected to a lot of vertices with the same attributes, while  $v_3$  is connected to vertices that do not have the same attributes as the neighbours of  $v_1$  and  $v_2$ . This means that, while the three vertices are all considered hubs,  $v_1$  and  $v_2$  should be considered more similar based on their connections, compared to  $v_3$ .

This also makes sense, since  $v_1$  and  $v_2$  are both elementary school teachers, while  $v_3$  is a university professor. Additionally,  $v_3$  should still be considered more similar to  $v_1$  and  $v_2$  than to any periphery vertex, since they are all hubs, i.e. teachers and not students. Similarity measures using attributes in combination with structure should not only be able to tell the difference between students and teachers, but also the different kinds of students and teachers, i.e. elementary school teachers and university professors.

### 3.4 Structural similarity in multirelational graphs

For multirelational graphs it can be more complex to define what the similarity between two vertices should be. For example, consider a social network modelling a workplace where there are two different relations: Co-worker and Friend. Two people, e.g. two managers, may be structurally similar(hubs) in the co-worker relation, while in the Friend relation one might socialize a lot with co-workers, while the other prefers separation between personal and professional life. This means that two vertices can be very similar in one relation but dissimilar in another.

This leads to an initial notion of how to define similarity in multirelational graphs, which is that two vertices are similar if they are similar in every relation. Figure 10 shows a multirelational graph with two relations. The two vertices  $v1$ (blue) and  $v5$ (orange) are maximally similar in both relations.

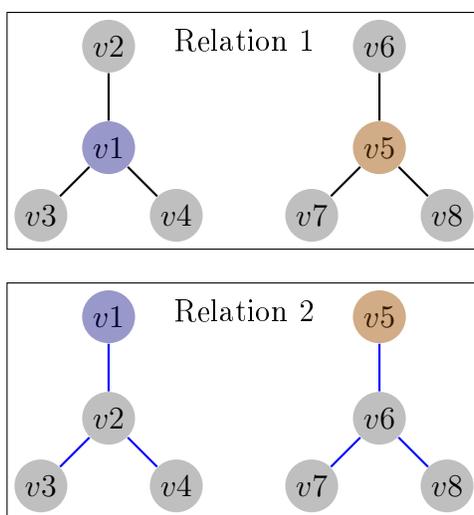


Figure 10: Multirelational graph (relations pictured separately)

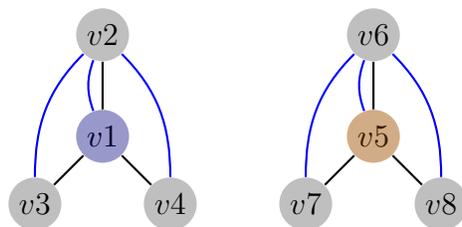


Figure 11: Multirelational graph

To obtain a single similarity score for  $v1$  and  $v5$ , the average similarity of the pair across all the relations could be considered.

Formally, let  $s$  be a vertex similarity measure for a single relational graph. Let  $G = (V, E, R)$  be a multirelational graph, and let  $u$  and  $v$  be two vertices in  $G$ . Then the multirelational version of  $s$ , denoted  $s_m$ , is defined in Equation 10. Here,  $s_i(u, v)$  is the similarity between  $u$  and  $v$  in relation  $i$ .

$$s_m = \frac{1}{|R|} \sum_{i=1}^{|R|} s_i(u, v) \quad (10)$$

It may be beneficial to look at the similarity score as a vector, where entry  $i$  is the similarity score for the two vertices in relation  $i$ , see Equation 11.

$$s_m(u, v) = (s_1(u, v), s_2(u, v), \dots, s_{|R|}(u, v)) \quad (11)$$

This simple notion on multirelational vertex similarity may not be appropriate in all cases. In Figure 12 we have a multirelational graph with two relations. In both relations,  $v1$ (blue) and  $v4$ (orange) are maximally similar. From our previous notion, these two vertices should be maximally similar.

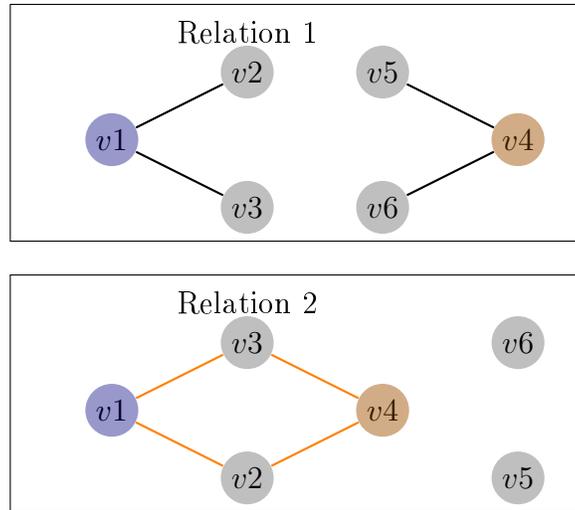


Figure 12: Multirelational graph (relations pictured separately)

However, the same graph is depicted in Figure 13 as a single graph. In this figure it can clearly be seen that the two vertices  $v1$ (blue) and  $v4$ (orange) are not maximally similar.

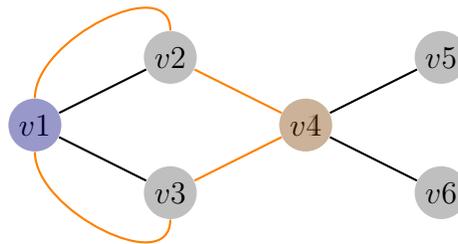


Figure 13: A multirelational graph

From this example it is clear that it can be too coarse to consider each relation as independent. The simplest way to deal with this is to convert the multirelational graph to a simple graph and then use a similarity measure designed for simple graphs. This is a good strategy if the relations themselves do not matter for the task at hand. However in many cases the information provided by the different relations should be utilized. As an example of how such a measure could be created, we extend the Jaccard similarity, defined in Equation 7. For a vertex  $v$ , let  $X(v) = \{(u, r) | (u, v, r) \in E\}$ , then the similarity between two vertices can be defined in Equation 3.4.

$$s_{mult-jaccard}(u, v) = \frac{|X(u) \cap X(v)|}{|X(u) \cup X(v)|} \quad (12)$$

### 3.4.1 Hubs in multirelational networks

For single relational networks, hubs are considered vertices which have a large degree relative to its neighbours. However, for multirelational networks it is a little less obvious what should be considered a hub. Some examples of possible hubs for multirelational networks are shown in Figure 14 figures.  $a$  is hub due to having a large degree relative to its neighbours. However, all edges belong only to a single relation.  $b$  is a hub for mostly the same reason as  $a$ , however, it is connected to all other vertices by two relations, and can be considered a hub in both the black and the green relation.  $c$  and  $d$  are more interesting, as they do not have a large degree in any one relation, but if we consider all relations at the same time both could be considered hubs.

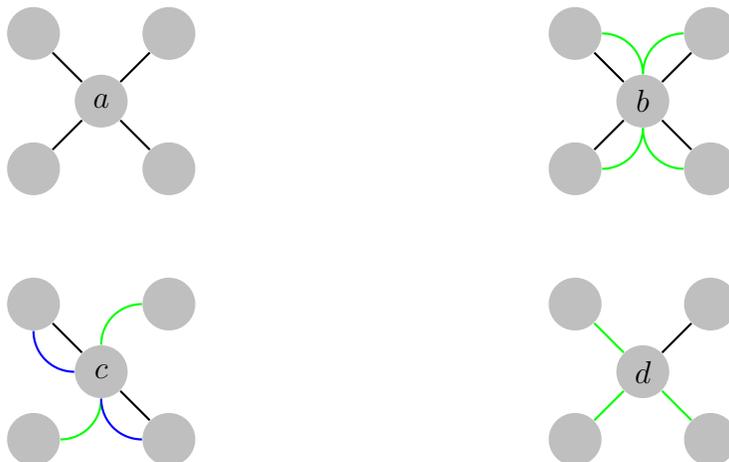


Figure 14: Four different multirelational graphs, with centres as hubs.

Being able to identify hubs in networks is of interest, because they play a central role in many real life networks. This includes protein-protein interaction networks, where the deletion of hub proteins are more likely to be lethal for an organism, compared to the deletion of lesser connected proteins[7].

## 4 Methods

Sections 4.1 to 4.4 (excluding 4.1.1) are largely based on last semesters report: *Vertex Similarity in Graphs*

Many similarity measures exist and they come in many forms. In the following sections we explore some of the more popular similarity measures for both graphs and vertices. We also introduce node2vec, a state of the art feature learning method, and show our extensions to it.

### 4.1 Isomorphism

The strictest type of similarity between two graphs is to check whether the two graphs are identical. When two graphs are identical we say that they are isomorphic. Formally two graphs  $G_1(V_1, E_1)$  and  $G_2(V_2, E_2)$  are isomorphic when there exist a bijection  $\mu : V_1 \rightarrow V_2$  such that  $\forall v_x, v_y \in V_1$  it holds that  $v_x \sim v_y$  iff  $\mu(v_x) \sim \mu(v_y)$ .

The definition of isomorphism is slightly different if the graph is labelled, as label preservation is required for the bijection.

Assume we have a similarity measure  $s_{iso}(G_1, G_2)$ , that checks whether two graphs are isomorphic. If they are isomorphic then  $s_{iso}(G_1, G_2) = 1$  and if they are not  $s_{iso}(G_1, G_2) = 0$ . An example of where  $s_{iso}(G_1, G_2)$  works is when you are only interested in whether two graphs are completely alike and all other cases are not important.

An obvious disadvantage of isomorphism is, that it is not possible to quantify how similar two graphs are, unless they are isomorphic. So if any data is missing or there is noise in the data, testing for isomorphism may not yield any useful results. For example in Figure 15 we can see that the two graphs are structurally similar, but  $s_{iso}(G_1, G_2) = 0$ .

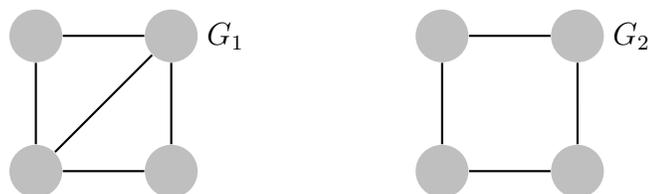


Figure 15: Two almost identical graphs

In Figure 16 we see two isomorphic graphs. They may not initially look isomorphic, but that is caused by how they are projected. We can see that they are isomorphic, because the two graphs have the same number of vertices, the same number of edges, and their degree sequences are the same. Furthermore, for any two vertices in  $G_3$ , if they are connected, they are also connected in  $G_4$  after being applied to the bijection  $\mu$ .

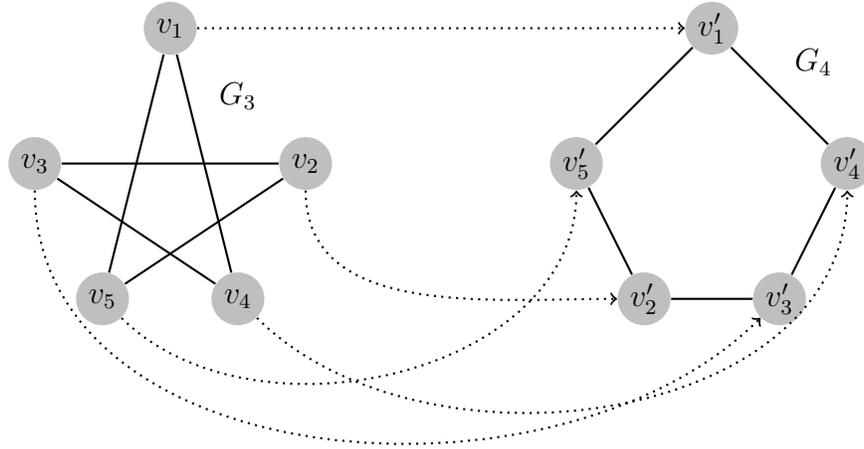


Figure 16: Two isomorphic graphs. Arrows indicate the mapping

#### 4.1.1 Automorphism

Closely related to isomorphism is automorphism, which is a graph being isomorphic to itself. Formally there exist an automorphism for a graph  $G(V, E)$  if there exist a bijection  $\mu : V \rightarrow V$  such that  $\forall v_x, v_y \in V$  it holds that  $v_x \sim v_y$  iff  $\mu(v_x) \sim \mu(v_y)$ . An example of an automorphism can be seen in Figure 17, where  $v_2$  and  $v_3$  are mapped to themselves,  $v_1$  is mapped to  $v_4$  and  $v_4$  is mapped to  $v_1$ . As with isomorphism it is also possible to create a similarity measure,  $s_{auto}(v_1, v_2)$ , for two vertices, where  $s_{auto}(v_1, v_2) = 1$  if there exist an automorphism where  $v_1$  is mapped to  $v_2$  and  $s_{auto}(v_1, v_2) = 0$  otherwise. This similarity measures has the same problem as  $s_{iso}$  in that it is too strict.

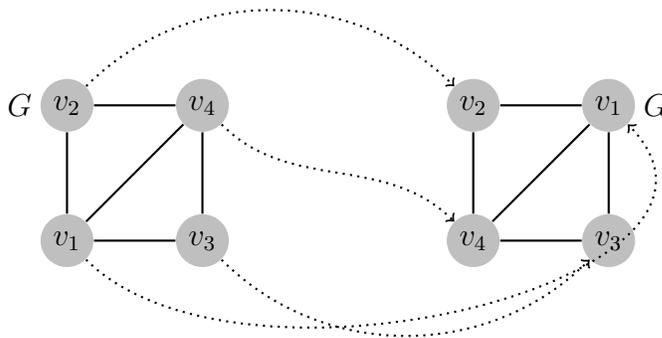


Figure 17: An automorphism of a graph. Arrows indicate the mapping

## 4.2 Graph edit distance

The Graph Edit Distance (GED) of two graphs,  $G_1$  and  $G_2$ , is a distance measure between two graphs. GED measures the distance between the two graphs  $G_1$  and  $G_2$  based on the minimum number of operations required to transform  $G_1$  into a graph which is isomorphic to  $G_2$ . GED is defined in Equation 13.

$$GED(G_1, G_2) = \min_{(o_1, \dots, o_k) \in P(G_1, G_2)} \sum_{i=1}^k c(o_i) \quad (13)$$

where  $P(G_1, G_2)$  is the set of edit paths transforming  $G_1$  into  $G_2$  and  $c(o) \geq 0$  is the cost of operation  $o$ .

Some common operations for transformation include:

- vertex insertions/deletions/substitutions
- edge insertions/deletions/substitutions
- label substitutions

The problem of computing the GED is NP-complete, thus for larger networks, it is too computational expensive.

## 4.3 Graph Kernels

A graph kernel  $k$  is a function  $k : X \times X \rightarrow \mathbb{R}$ , where  $X$  is either a set of graphs, vertices or edges. The function has to fulfill two criteria to be considered a kernel. Firstly the kernel has to be symmetric i.e.  $k(x_1, x_2) = k(x_2, x_1)$ . Secondly the kernel must be positive definitive, i.e.  $\sum_{i=1}^n \sum_{j=1}^n c_i c_j k(x_i, x_j)$  has to be positive for any  $(c_1, c_2, \dots, c_n) \in \mathbb{R}$  and  $(x_1, x_2, \dots, x_n) \in X$ . A kernel can often be thought of as a similarity measure.

One problem these kernels share, is that some graph is not necessarily maximally similar to itself. This is because the similarity score is obtained by counting. This has the unfortunate downside of allowing a graph to be more similar to some other graph than to itself, simply because the other graph might be larger. To solve this problem, as with similarity measures previously mentioned, kernels are normalized to be in the range of zero to one. This normalization can be done using Equation 14.

$$k'(x, y) = \frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}} \quad (14)$$

Additionally, we use graph similarity measures as vertex similarity measures. This can be done by considering the  $k$ -neighbourhoods of two vertices. Let  $N_k(v_1) \subseteq G_1$  and  $N_k(v_2) \subseteq G_2$  be the  $k$ -neighbourhood subgraphs induced by  $v_1 \in V(G_1)$  and  $v_2 \in V(G_2)$  respectively. The similarity of  $v_1$  and  $v_2$  is then  $k_{sim}(N_k(v_1), N_k(v_2))$ , where  $k_{sim}$  is a graph similarity measure.

The following subsections show some popular graph kernels, namely the Marginalized kernel and the Shortest Path kernel.

### 4.3.1 Marginalized Graph Kernel

One example of a graph kernel, explained in Tsuda et al.[18], is the marginalized graph kernel(MG), which compares random walks in two attributed graphs. An example of different random walks in a graph can be seen in Figure 18. In this example only vertices have attributes, but if the edges have attributes these will need to be included in the walks.

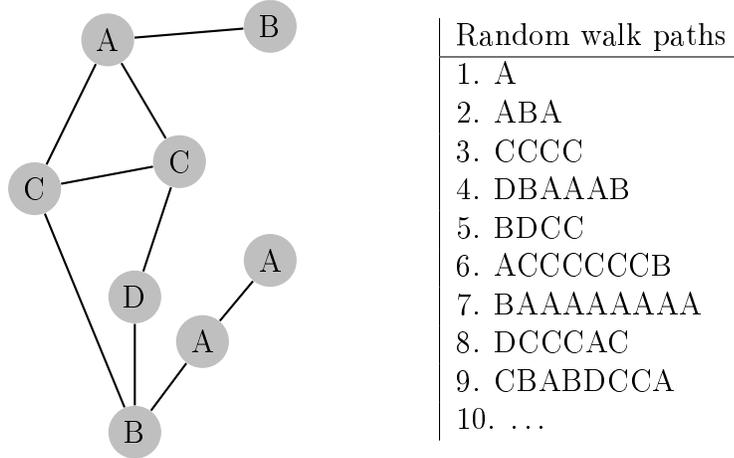


Figure 18: Example of random walks on a graph

A random walk  $h$  is created by first choosing a starting vertex  $v_{h1}$  with probability  $p_{start}(v_{h1})$ . When the walk is on vertex  $v_{hi}$  it may continue to vertex  $v_{hj}$  with probability  $p_t(v_{hj}|v_{hi})$  or end the walk with probability  $p_{stop}(v_{hi})$ .

The probability for an entire random walk  $h$  is given in Equation 15:

$$p(h|G) = p_{start}(v_{h1})p_t(v_{h2}|v_{h1}) \dots p_t(v_{hn}|v_{hn-1})p_{stop}(v_{hn}) \quad (15)$$

The start, transition and stop probabilities depend on the application domain, but if no domain knowledge is available it is common to assume uniform probability distributions for start and stop probabilities. For transition probabilities the distribution for a vertex is uniform only over its neighbours.

MG measures the similarity of two graphs based on the random walks that can be created in the graphs, and is formally defined in Equation 16.

$$k_{mg}(G, G') = \sum_h \sum_{h'} k_{walk}(h, h')p(h|G)p(h'|G') \quad (16)$$

where  $h$  and  $h'$  are walks in graph  $G$  and  $G'$  respectively,  $p(h|G)$  is the probability that walk  $h$  occur in graph  $G$ , defined in Equation 15, and  $k_{walk}(h, h')$  is a kernel that compares the two walks.  $k_{walk}(h, h')$  is defined in Equation 17.

$$k_{walk}(h, h') = \begin{cases} k_v(v_{h1}, v_{h'1}) \prod_{i=2}^{|h|} k_e(e_{hi-1}, e_{h'i-1})k_v(v_{hi}, v_{h'i}), & \text{if } |h| = |h'| \\ 0 & \text{if } |h| \neq |h'| \end{cases} \quad (17)$$

Here  $|h|$  is the length of walk  $h$ ,  $v_{hi}$  is the  $i$ 'th vertex in  $h$ ,  $e_{hi}$  is the  $i$ 'th edge in  $h$ , and  $k_v$  and  $k_e$  are kernels that compare vertices and edges respectively. An example of a vertex kernel, which could be used in case of a single discrete vertex attribute  $\alpha$  could be the identity kernel:

$$k_{id}(v, v') = \begin{cases} 1 & \text{if } \alpha(v) = \alpha(v') \\ 0 & \text{otherwise} \end{cases} \quad (18)$$

In case of a single continuous vertex attribute  $\alpha$  the Gaussian kernel could be used:

$$k_{gau}(v, v') = \exp\left(\frac{-\|\alpha(v) - \alpha(v')\|^2}{2\sigma^2}\right) \quad (19)$$

The identity kernel or Gaussian kernel can also be used as edge kernel. If a graph has no edge attributes, the edge kernel can be defined to always return one.

A problem with kernels based on random walks is that in some cases they can not differentiate between graphs with different structures. This case is illustrated in Figure 19. In these graphs all label attributes on the vertices are the same, so the random walks generated in the two graphs would always be the same. This results in the normalized marginalized kernel returning one (maximum similarity) when it compares the two graphs. However, as can be seen, the graphs are structurally dissimilar.



Figure 19: Two graphs that are structurally dissimilar but receives maximum similarity by MG

### 4.3.2 Marginalized Vertex Kernel

Another kernel based on random walks is the marginalized vertex kernel(MV). Li et al [11] shows how this kernel can be used to predict gene functions in a gene interaction networks. MV works similarly to MG, but instead of comparing graphs it compares vertices. Let  $v$  and  $v'$  be vertices in graph  $G$ . MV is defined in Equation 20.

$$k_{mv}(v, v') = \sum_h \sum_{h'} k_{walk}(h, h')p(h|G)p(h'|G) \quad (20)$$

where  $h$  and  $h'$  are walks starting from  $v$  and  $v'$  respectively.  $p(h|G)$  is the probability that walk  $h$  occur in graph  $G$ , shown in Equation 15, with the starting probabilities for  $v$  and  $v'$  set to one, and set to zero for all other vertices.  $k_{walk}(h, h')$  is a kernel that compares the two walks, and is defined in Equation 17.

MV can be efficiently computed using matrices. This allows you to use optimized matrix operations to compute the kernel values of all vertex pairs at the same time. Let  $M = \{M_{i,j}\} = \{p_t(j|i)\}$  be the transition matrix and  $Q = \{Q_{i,j}\} = \{p_{stop}(j)\}$  be the stopping probability matrix. Here,  $p_t(j|i)$  is the probability that a walk transitions to vertex  $j$  given that it is in vertex  $i$ , and  $p_{stop}(j)$  is the probability that a walk stops in vertex  $j$ . The kernel is then defined in Equation 21.

$$K = \sum_{i=1}^l K_i, \quad K_1 = (M * Q)K_v(M * Q)^T, \quad K_{i+1} = M(K_v * K_i)M^T \quad (21)$$

where  $K = \{k_{mv}(v_i, v_j)\}$  is the MV kernel matrix,  $K_v = \{k_v(i, j)\}$  is the vertex kernel matrix,  $l$  is the maximum walk length, and  $*$  is the Hadamard product. In this equation,  $K_i$  represents walks of length  $i$ .

MV has the same limitation as MG regarding graph structure.

### 4.3.3 Shortest Path Graph kernel

The shortest path graph kernel (SPG)[4] compares two graphs using shortest path transformations. It starts by transforming the graphs compared into so called shortest path graphs, and then comparing these based on their edges. A shortest path graph  $sG$  of the graph  $G$  has every vertex of  $G$ , and  $sG$  has an edge between every vertex that are connected by a path in  $G$ . The weight of an edge  $(v_i, v_j)$  in  $sG$  is the length of the shortest path between  $v_i$  and  $v_j$  in  $G$ .

An example of a shortest path transformation can be seen in Figure 20 and Figure 21. Here it can be seen, for example, that the shortest path between vertex  $a$  and vertex  $d$  is of length four, so because of this, in the shortest path graph there is an edge between  $a$  and  $d$  with weight four.

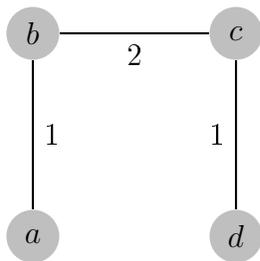


Figure 20: Original graph  $G$

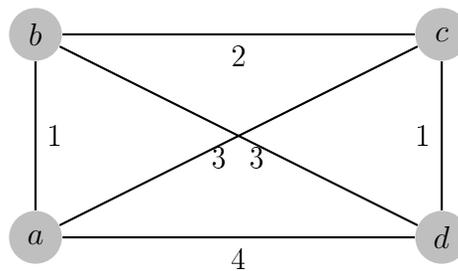


Figure 21: Shortest path transformation  $sG$

Let  $sG$  and  $sG'$  be the shortest path transformations of the graphs  $G$  and  $G'$ , then SPG is formally defined in Equation 22

$$k_{spg}(sG, sG') = \sum_{e \in E(sG)} \sum_{e' \in E(sG')} k_{walk}(e, e') \quad (22)$$

where  $k_{walk}(e_i, e_j)$  is the previously introduced walk kernel in Equation 17, in which the edges  $e_i$  and  $e_j$  are considered walks of length 2. For example, if  $e_i = (v_i, v_j)$ , then  $k_{walk}(e_i, e_j)$  considers  $e_i$  to be a walk of the form  $v_i, e_i, v_j$ .

#### 4.3.4 Shortest Path Vertex kernel

While SPG compares two graphs, we are interested in a measure for measuring the similarity between two vertices. We propose the shortest path vertex kernel (SPV) which is based on the concepts used by SPG. This similarity measure compares two vertices. The first step to SPV is to create shortest path stars for the two vertices that need to be compared. The shortest path star  $s_*v$  for a vertex  $v \in V(G)$  contains all the vertices from  $G$  that are connected with a path to  $v$ . There is an edge between  $v$  and every other vertex in  $s_*v$ . The weight of an edge  $(v, v')$  in  $s_*v$  is the length of the shortest path between  $v$  and  $v'$  in  $G$ . Figure 22 shows a graph and Figure 23 its shortest path star transformation for the vertex labelled  $a$ .

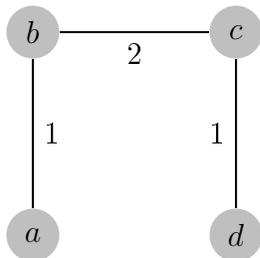


Figure 22: Original graph

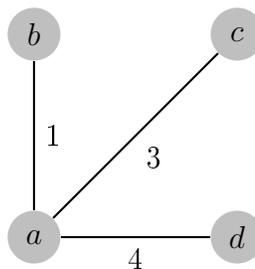


Figure 23: Shortest path star transformation

Once the shortest path stars have been created, SPV can be defined exactly as SPG, see Equation 22, with the exception that the shortest path stars are used instead of  $sG$  and  $sG'$ .

## 4.4 Feature based methods

Another approach to measuring the similarity between two graphs, is to transform them into feature vectors. Subsequently, a similarity measure that is based on vectors, such as the dot product, can be used. Features can, for example, be based on paths or frequent substructures.

Features can also be used for labelling unlabelled graphs, allowing for graph similarity measures that require labelled graphs to be used on unlabelled graphs. Examples of such approaches are the graphlet kernels[17] and the Weisfeiler-Lehman Subtree Kernel[16]. Another useful aspect of features is that they allow for indexing graphs based on the features they have[19]. This can help speed up information retrieval in large graph databases, as you are able to filter out graphs that does not contain these features.

#### 4.4.1 Weisfeiler-Lehman Subtree Kernel

The Weisfeiler-Lehman Subtree Kernel (WL)[16] is based on the Weisfeiler-Lehman test of isomorphism between two graphs. The isomorphism test works in iterations, where each iteration updates the label attribute of each vertex based on the labels of their neighbours. It runs until the label sets of the two graphs differ, or it has run  $n$  iterations and the two graphs still have the same label set. The two graphs are not isomorphic if their label sets differ. Unlike the isomorphism test, the subtree kernel doesn't stop when the two label sets differ, but instead runs the full  $n$  iterations. It generates a feature vector for each graph which counts the labels of the graph, and outputs the dot product between these. The algorithm works as follows:

- (a) WL takes two labelled graphs as input (see Figure 24)
- (b) Each vertex label is represented as a multiset of labels, and the labels of its neighbours are added in sorted order (see Figure 25)
- (c) Each multiset of labels is mapped to a new label with an injective mapping. We map to labels by going through the multisets in sorted order, and incrementing the label whenever a new multiset is encountered (see Figure 26)
- (d) The labels of the graphs are updated to the new labels created in step (c) (see Figure 27)
- (e) After  $n$  iterations we have a feature vector  $\phi$  for each graph, which counts how many of each label (both original from Figure 24 and updated ones from Figure 27) each graph has. The kernel outputs the dot product of these feature vectors (see Figure 28, the counts are of the labels in sorted order, where the first five are the original labels)

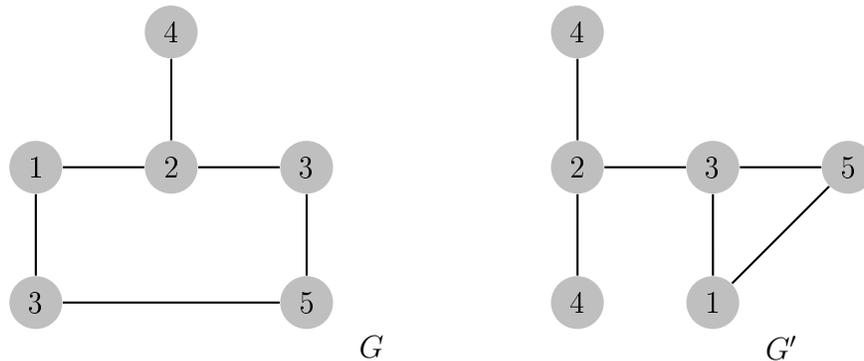


Figure 24: Given labelled graphs G and G'

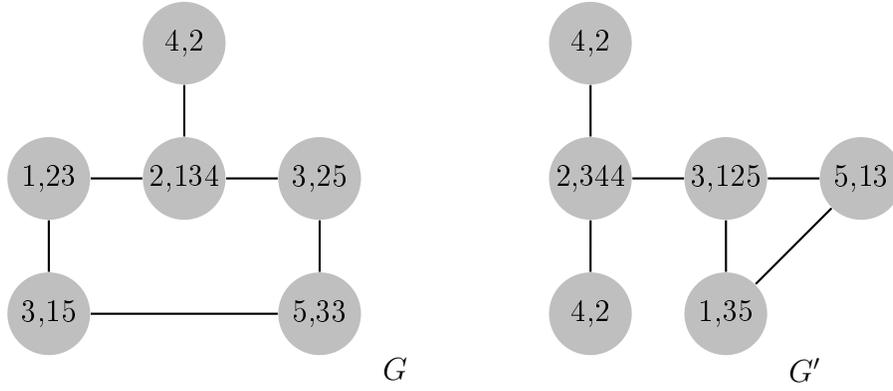


Figure 25: Neighbour labels added to the label multiset after one iteration

1,23	→	6	3,15	→	11
1,35	→	7	3,25	→	12
2,134	→	8	4,2	→	13
2,344	→	9	5,13	→	14
3,125	→	10	5,33	→	15

Figure 26: Compressed labels after one iteration

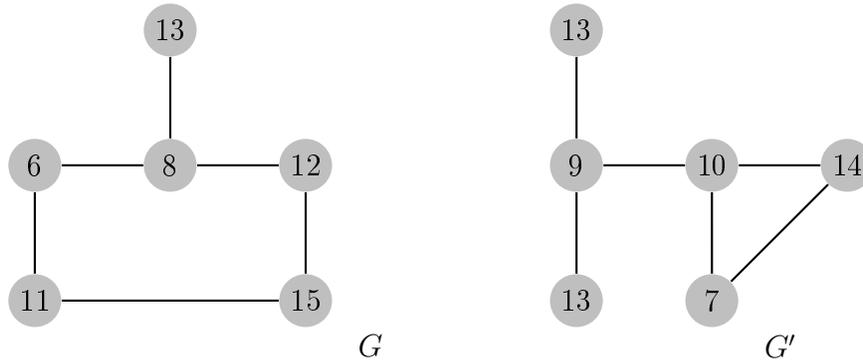


Figure 27: Relabelled graph after one iteration

$$\begin{aligned} \phi(G) &= (1, 1, 2, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1) \\ \phi(G') &= (1, 1, 1, 2, 1, 0, 1, 0, 1, 1, 0, 0, 2, 1, 0) \\ k_{wl}(G, G') &= \langle \phi(G), \phi(G') \rangle = 9 \end{aligned}$$

Figure 28: Feature vectors after one iteration

WL is faster than most other graph kernels, and works well for use cases where isomorphism is important. Because the kernel uses discrete vertex labels it is only useful on data where it is possible to create meaningful discrete labels.

## 4.5 Feature learning

Recently, methods have been proposed which use feature learning techniques, from natural language processing(NLP), to create embeddings of vertices. Specifically, they create random walks which are used as input to the Skip-gram framework[15][14]. In this section we will explain the Skip-gram model as well as node2vec[6], one of the newest feature learning methods for vertices. We will also detail how to extend node2vec, such that it is able to utilize vertex and edge attributes.

### 4.5.1 Skip-gram model

Skip-gram is a probabilistic model used in the NLP domain for learning embeddings of words[14][15]. Often NLP models aim to estimate the likelihood of observing a word,  $w_t$ , given some context, i.e.  $P(w_t|w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c})$  where  $\{w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c}\}$  is the context. However, Skip-gram aims to do the opposite, that is, predict the context given a word. This is accomplished by maximizing the following log-likelihood objective function, see Equation 23.

$$\sum_{t=1}^T \log P(w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c}|w_t) \quad (23)$$

$T$  is the text corpus. Larger sizes of  $c$  can result in more accurate predictions at the cost of training time. The probability  $P(w_{t-c}, \dots, w_{t+c}|w_t)$  is computed as shown in Equation 24.

$$\prod_{-c \leq j \leq c, j \neq 0} P(w_{t+j}|w_t) \quad (24)$$

It is assumed that the words in the context are conditional independent given the target word  $w_t$ . The equivalent to Equation 24 using log is:

$$\sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j}|w_t) \quad (25)$$

The probability  $P(w_{t+j}|w_t)$  is often defined using the softmax function as defined in Equation 26.

$$P(w_o|w_i) = \frac{\exp(v_{w_o}^\top v_{w_i})}{\sum_{w=1}^W \exp(v_w^\top v_{w_i})} \quad (26)$$

$w_o$  correspond to the context word and  $w_i$  to the target word. Here  $v_w$  and  $v'_w$  are input and output embeddings for the word  $w$ , and  $W$  is the vocabulary. As Skip-gram only takes as input two words at a time, input(target)/output(context), one can view

the training data to consist of not sentences, but context-target pairs created from these sentences. For example, consider the sentence "The dog barks" with context size being one. Then two context-target pairs are created with "dog" as the target word are (the, dog) and (barks, dog).

One problem with Skip-gram is that using softmax is often impractical due to its computational cost being proportional to  $W$ , which can be quite large in some cases. Instead, either hierarchical softmax or negative sampling is used for training the skip-gram model.

**Hierarchical softmax** Hierarchical softmax is an efficient approximation of the softmax function shown in Equation 26. Hierarchical softmax assign output words to the leaves of a binary tree, where every vertex represents the relative probabilities of its children. Using this representation the prediction problem now becomes to maximize the probability of paths from the root to leaves.

More formally let  $B(w_o) = \{b_0, b_1, \dots, w\}$  be the path from the root vertex( $b_0$ ) to the word  $w_o$ . In addition, for any inner vertex  $b$  let  $ch(b)$  be a random child vertex of  $b$ , and let  $[x]$  be one if  $x$  is true and minus one otherwise. Then Hierarchical softmax defines the probability  $P(w_o|w_i)$  as in Equation 27.

$$P(w_o|w_i) = \prod_{j=1}^{|B(w_o)|-1} \sigma([b_{j+1} = ch(b_j)])v'_{b_j} v_{w_i}^T \quad (27)$$

Sigma is defined as  $\sigma(x) = \frac{1}{1+\exp(-x)}$ .  $v_{b_j}$ , which is the vector representation of the inner vertices in the binary tree.

The cost of computing the Hierarchical softmax is proportional to  $\log(W)$  instead of  $W$ .

**Negative sampling** Instead of using hierarchical softmax one can use negative sampling, which is a model for differentiating data from noise utilizing logistic regression. One way to define negative sampling, is shown in Equation 28.

$$\log \sigma(v_{w_o}^T v_{w_i}) + \sum_{I=1}^k \mathbb{E}_{w_I \sim P_n(w)} \log(\sigma(-v_{w_I}^T v_{w_i})) \quad (28)$$

Equation 28 replaces all instances of  $\log P(w_o|w_i)$  in Equation 25.

The task of negative sampling is to learn the difference between the target word  $w_o$  and  $k$  samples drawn from a noise distribution  $P_n(w)$  via logistic regression. The samples from the noise distribution are assumed to be negative i.e. they do not appear in the same context as the input word  $w_i$ . Mikolov et al.[15] estimate that for small datasets an appropriate value for  $k$  is 5-20, but for larger datasets  $k$  can be 2 – 5.

The distribution  $P_n(w)$  which negative samples are drawn from is a free parameter. Mikolov et al.[15] found that the best distribution was the unigram distribution( $U(w)$ ) raised to the 3/4th power. For a more formal definition see 29, where  $U(w) = \frac{freq(w)}{|T|}$  and  $Z$  is a normalizations constant.

$$P_n(w) = \frac{U(w)^{\frac{3}{4}}}{Z} \quad (29)$$

**Frequent word subsampling** It can be common in text corpora, that there are words which appear with a very high frequency. This is usually words such as "the", "a" etc.. These words often provide less information than less common words, for example, in the sentence "the dog barks", "dog" and "barks" is a more meaningful co-occurrence than "the" and "dog". In addition, the feature vectors for frequent words do not change much after many million occurrences. Therefore skip-gram employ subsampling of frequent words. Basically, when learning there is a certain probability for a word being kept in the corpus, which is defined in Equation 30<sup>1</sup>.

$$P(w_i) = \left( \sqrt{\frac{U(w_i)}{0.001}} + 1 \right) \cdot \frac{0.001}{U(w_i)} \quad (30)$$

Here  $P(w_i)$  is the probability of word  $w_i$  being kept, and  $U(w_i)$  is the frequency of  $w_i$ .

#### 4.5.2 Node2vec

Node2vec is an example of feature learning by Grover et al.[6]. Node2vec is a framework for learning vertex embeddings. Practically, this means learning a mapping of vertices to euclidean space that maximizes the likelihood of preserving network neighbourhoods of vertices. Formally they define the objective function seen in Equation 31.

$$\max_f \sum_{u \in V} \log P(N_S(u) | f(u)) \quad (31)$$

Here  $u$  is a vertex,  $N_S(u)$  is the network neighbourhood of vertex  $u$  generated by sampling strategy  $S$  and  $f(u)$  is the embedding of  $u$ .

This objective function is very similar to the Skip-gram objective function, where  $f(u)$  is the input word and  $N_S(u)$  is the context, so to solve this function the authors use Skip-gram. There is, however, one problem which needs to be solved, before Skip-gram can be applied, and that is to determine how the neighbourhood should be represented.

They take influence from two sampling strategies, namely, Breadth-first sampling(BFS) and Depth-first sampling(DFS). BFS samples the immediate neighbourhood of the source vertex, and DFS samples vertices which are increasingly far away from the source vertex. According to the authors, vertices sampled by BFS closely corresponds to structural similarity, meaning roles such as a hubs in networks can be inferred using this strategy. Vertices sampled by DFS reflect proximity. The authors do not directly use these sampling strategies, but instead sample  $2^{nd}$  order biased

---

<sup>1</sup>This is the probability used in the implementation. Mikolov et al.[15] uses a different definition

random walks, which can be configured to behave like either BFS, DFS or a mix of both.

These  $2^{nd}$  order random walks are used to represent the neighbourhood of a vertex. The walks are configurable by the parameters  $p$  and  $q$ .  $p$  affects the likelihood of revisiting vertices in the walk. A high  $p$  value ( $> \max(q, 1)$ ) ensures that already sampled vertices are less likely to be sampled again, which encourages moderate exploration of the network in the veins of DFS. A low  $p$  value ( $< \min(q, 1)$ ) leads to a lot of backtracking, keeping the walk local to the source vertex like BFS would.  $q$  affects whether the walk is biased towards visiting vertices that are close or far away from the start vertex. If  $q > 1$  the walk is biased towards vertices close to the source vertex, approximating BFS behaviour. If  $q < 1$ , the walk is biased towards visiting vertices far away from the source vertex, reflecting a DFS-like exploration. The unnormalized transition probability of the biased random walk,  $\pi_{vx}$ , is defined as  $\alpha_{pq}(t, x) \cdot w_{vx}$ , where  $v, x, t \in V$  and  $v$  is the current vertex,  $t$  is the previous vertex and  $x$  is the next vertex. Here  $\alpha_{pq}(t, x)$  is the search bias defined by  $p$  and  $q$  as seen in Equation 32, and  $w_{vx}$  is the weight on edge  $(v, x)$ . For unweighted graphs  $w_{vx} = 1$ .

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } dist_{tx} = 0 \\ 1 & \text{if } dist_{tx} = 1 \\ \frac{1}{q} & \text{if } dist_{tx} = 2 \end{cases} \quad (32)$$

Here  $dist_{tx}$  is the shortest path distance from  $t$  to  $x$ . An example of this is shown in Figure 29, where the walk has just traversed edge  $(t, v)$  and are now stood at vertex  $v$ . We can see that lower values of  $p$  encourages the walk to backtrack and lower values of  $q$  leads to exploration away from the source vertex.

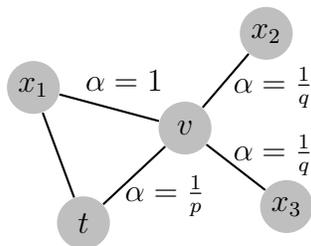


Figure 29: An example of the biased random walk where edges are labelled with search bias

In Figure 30 the implementation of node2vec is shown. We can see that node2vec has six parameters in total, with  $p$  and  $q$  being two of them. Additional parameters include number of dimensions  $d$ , number of walks  $r$ , length of the walks  $l$  and size of the context  $k$ .

---

**Algorithm 1** The *node2vec* algorithm.

---

**LearnFeatures** (Graph  $G = (V, E, W)$ , Dimensions  $d$ , Walks per node  $r$ , Walk length  $l$ , Context size  $k$ , Return  $p$ , In-out  $q$ )  
 $\pi = \text{PreprocessModifiedWeights}(G, p, q)$   
 $G' = (V, E, \pi)$   
Initialize *walks* to Empty  
**for** *iter* = 1 **to**  $r$  **do**  
    **for all** nodes  $u \in V$  **do**  
         $walk = \text{node2vecWalk}(G', u, l)$   
        Append *walk* to *walks*  
 $f = \text{StochasticGradientDescent}(k, d, \textit{walks})$   
**return**  $f$

---

**node2vecWalk** (Graph  $G' = (V, E, \pi)$ , Start node  $u$ , Length  $l$ )  
Initialize *walk* to  $[u]$   
**for** *walk\_iter* = 1 **to**  $l$  **do**  
     $curr = \textit{walk}[-1]$   
     $V_{curr} = \text{GetNeighbors}(curr, G')$   
     $s = \text{AliasSample}(V_{curr}, \pi)$   
    Append  $s$  to *walk*  
**return** *walk*

---

Figure 30: node2vec psudocode as seen in their paper[6]

node2vec consists of three phases, namely, preprocessing, walk simulations and optimization. In the preprocessing phase, the transition probabilities are computed according to  $p$  and  $q$ . In the walk simulations phase, random biased walks are done  $r$  times for each vertex. The optimization phase is where the objective function in Equation 31, i.e. Skip-gram, is maximized. The maximization is done using stochastic gradient descent, and the final results  $f$  is a resulting vertex embedding of size  $d$  for every vertex. For the Skip-gram part, node2vec uses negative sampling by default.

One drawback of node2vec is that it only uses the identifiers of vertices, thereby ignoring information such as vertex attributes or relation types. In the following sections we present extensions to the node2vec framework, which makes it possible to use this additional information.

### 4.5.3 node2vec for attributed graphs

In node2vec a random walk is a sequence of vertices i.e.

$$walk = \{v_1, v_2, \dots, v_n\}$$

where the vertices are represented by an identifier. In attributed graphs the attributes can provide valuable information. We propose capturing this information by still considering a walk as a sequence of vertices, but instead of a vertex being represented by an identifier, it is represented by a feature vector i.e.

$$walk = \{\phi(v_1), \phi(v_2), \dots, \phi(v_n)\}$$

where  $\phi(v_n)$  is the feature vector for vertex  $v_n$ . Liu et al. [12] represents vertices in a similar fashion.

A problem that now arises is what should these feature vectors contain. This highly depends on the kind of network or graph that one is working with.

In case of an attributed graph with only one categorical attribute, a simple feature vector for a vertex could consist of a vertex id and the attribute of the vertex.

As an example consider the graph in Figure 31 where the attribute of a vertex is shown as the label.

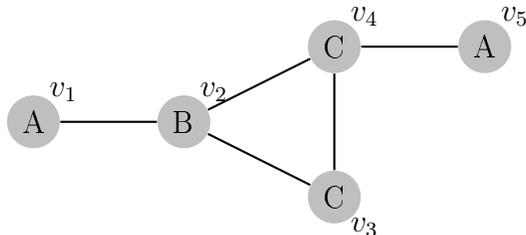


Figure 31: A graph labelled with an attribute

The random walk

$$\{v1, v2, v4, v5\} \tag{33}$$

will then be

$$\{v1, A, v2, B, v4, C, v5, A\} \tag{34}$$

This extension we refer to as *node2vec-attr*, and its feature vector looks as follows:

$$\phi_{attr}(v) = id, attribute$$

The expectation behind adding the attribute information to the walks, is that vertices who often appear with the same attributes should have embeddings which are closer together, than vertices that appear with different vertex attributes.

Note that as the vertex attributes are included in the walks, this means that Skip-gram will create embeddings for the attributes. These might provide some interesting information. For example, in networks where vertices with one attribute often have neighbours of the same attribute, it would be expected that the attribute embeddings should be close to these vertices.

With this representation there are three different kind of context-target pairs which are be created: vertex-vertex, vertex-attribute, and attribute-attribute pair. Note that with the vertex-attribute pair, both vertices and attributes can appear as context and target. Dependent on the task one might consider only using one or two of these kinds of pairs, for example, if one is only interested in the relationship between vertices and attributes the two other kinds of pair can be removed.

One problem that could arise from including attributes is that it becomes more difficult to learn the structural or proximity equivalences i.e. all that it might learn, is that a vertex often appears with an attribute or attributes.

As we are particularly interested in structural and semantic similarity, we introduce another extension, referred to as *node2vec-attr+*. We propose two vertex features for this extension, which could assist in learning these similarities.

The first feature is the sorted multi set of neighbour attributes, which we call *all-neighbours*. This is inspired by the Weisfeiler-Lehman kernel. This feature captures information not only on the neighbour attributes, but also on the degree of a vertex. This feature is strict as it will only be same for vertices with the same degree and with neighbours with the same attribute. For  $v_2$  in Figure 31 this feature will be  $ACC$ .

The second feature is very related as it is simply the sorted set of neighbour attributes. We call this feature *unique-neighbours*. This feature is a less strict version of the first all-neighbours. For  $v_2$  this feature is will be  $AC$ .

The feature vector for node2vec-attr+ will look as follows:

$$\phi_{attr+}(v) = id, attribute, all-neighbours, unique-neighbours$$

Adding extra features to the walks naturally increases the length of the random walk. Let  $l$  be the walk length parameter used by node2vec and let  $|\phi(v)|$  be the length of the feature vector, the final walk length  $fl$  is then:

$$fl = l \cdot |\phi(v)| \tag{35}$$

If the graphs vertices have several attributes, the feature representation of a vertex, could simply be all the attributes of the vertex. One problem can arise, if any of the vertex attributes are numeric, because Skip-gram were designed for words i.e. discrete inputs.

#### 4.5.4 Multirelational graphs

Up to now we assumed the graph operated on by node2vec were not multirelational. In non multirelational graphs there is no need to include what edge was taken in a random walk as this is implicit. The same can not be said for random walks in multirelational graphs. Consider the graph in Figure 32 and the random walk  $v_1, v_2, v_4, v_5$ . This walk can be created in several different ways, for example, to go from  $v_1$  to  $v_2$  the  $R_1$  or  $R_2$  edge could be taken.

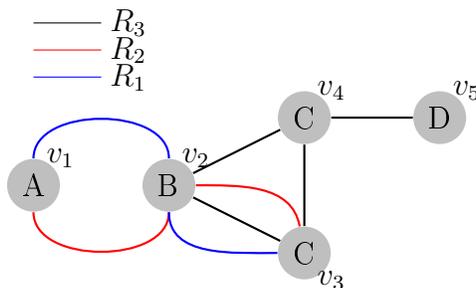


Figure 32: Multirelational graph

The simplest approach to a multidimensional extension, which we call *node2vec-multi*, is to include the specific edge taken in the feature vector of the vertex, i.e.

$$\phi_{multi}(v) = (id, relation)$$

So the walk from earlier could be  $\{v1, R1, v2, R3, v4, R3, v5\}$ .

## 5 Similarity measure criteria

*This section includes some material from last semesters report: Vertex Similarity in Graphs*

When deciding which similarity measures to use for a given problem, there are some criteria that should be examined. These criteria provide a quick overview of which similarity measures can be applied to datasets with different properties. I.e. if a similarity measure can only handle discrete data, and the dataset contains continuous data, the measure can quickly be discarded unless some manipulation is done to the data. The categories of the criteria we consider are: Connectivity, Local/Global, Discrete/Continuous, Multi-attributed, Multi-relational.

**Connectivity:** The connectivity criteria states that in order to have a similarity of above the minimal(zero in most cases) between two vertices, they need to be connected by a path.

Consider the case of having two isolated isomorphic graphs, as illustrated in Figure 33. When comparing any two vertices from different graphs, as there is no path between any vertices of the two graphs, their similarity will always be minimal using any similarity measure that requires connectivity. For example, a measure such as the Jaccard coefficient(see Equation 7) would not be able to provide any meaningful similarity between two vertices in different graphs, as they would never share any common neighbours, see Equation 36 and Equation 37.

$$s_{jaccard}(a, a') = \frac{0}{4} = 0 \quad (36)$$

$$s_{jaccard}(a, d) = \frac{1}{3} = 0.33 \quad (37)$$

While  $a$  and  $a'$  clearly look more similar than  $a$  and  $d$ , similarity measures that require connectivity will not be able to express this.

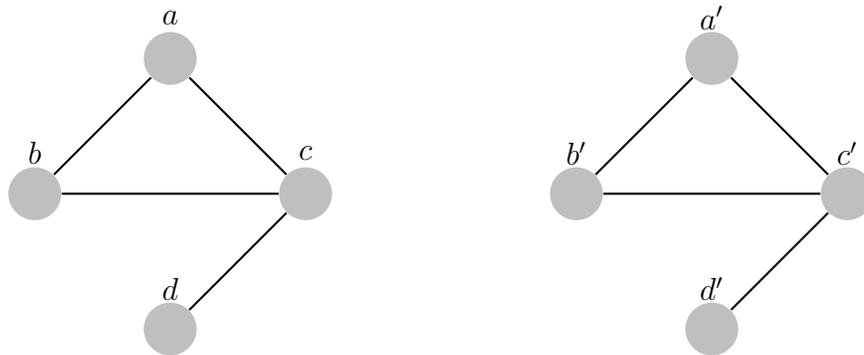


Figure 33: Example of two disconnected isomorphic graphs

This becomes relevant if we for example have a network modelling a university, where the different vertices correspond to the personnel of the university, students of the university, etc.. Let the two graphs in Figure 33 be two universities where in the left network  $a$  is a supervisor for students  $b$  and  $c$ . Using a similarity measure we would want to find a vertex(person) in the right network, which served a similar role(supervisor). As demonstrated earlier, a measure such as the Jaccard coefficient is unable to do this, whereas measures that does not require connectivity will be able to provide some similarity.

**Local/global:** In similarity measures for which the local criteria holds, only the intermediate neighbourhood(or  $k$ -neighbourhood for small  $k$ ) of each given vertex are considered when determining the similarity. Similarity measures for which the global criteria holds, all vertices are considered when determining the similarity.

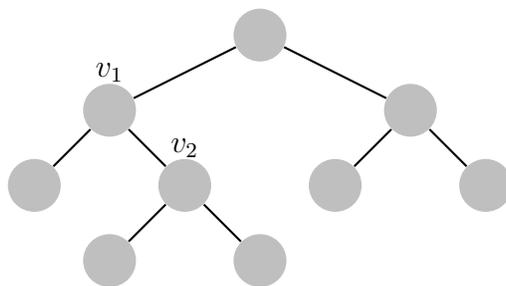


Figure 34: Example tree network

If we consider the tree-like network shown in Figure 34, we can see the difference between the local and global criteria. If we for example only consider the immediate neighbourhood of  $v_1$  and  $v_2$  they would be automorphic. However, from a global perspective, this is not the case, here  $v_1$  and  $v_2$  are seen as being different. Looking at their closeness centrality, we can also see that they are different, with  $C(v_1) = 0.60$  and  $C(v_2) = 0.50$ .

**Discrete/Continuous:** The discrete/continuous criteria states whether a measure works on discrete or continuous attributes. This is important to understand as it determines what network data it can be applied to, and real world networks often contain both.

**Attributed:** The attributed criteria states whether a measure can handle network data with attributes.

Networks often contain many different attributes, such as age and interests for social networks or keywords and publication date for citation networks. For certain problems, utilizing this data in combination with the structure of the network is crucial. For example, if we consider the two citation networks in Figure 35 and Figure 36 respectively. If using a measure which only looks at the structure, it would report  $v_1$  and  $v_2$  as being maximally similar. However, if we look at the attributes, we can see that they differ. Perhaps these attributes describe the domain in which these papers has been published, one in physics, one in biology. Depending on the task, you might have wanted to find similar papers to  $v_1$ , which is a physics papers.

Thus looking only at the structure in this case would not yield a satisfactory result, where a combination of structure and attributes might have.

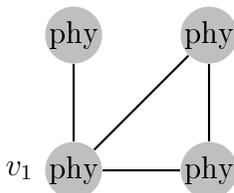


Figure 35: Citation network for  $v_1$

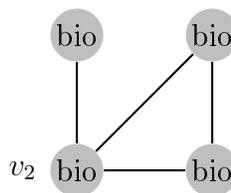


Figure 36: Citation network for  $v_2$

**Multi-relational:** Lastly, the multi-relational criteria states whether a measure can handle different types of relationships between vertices.

We choose these criteria as they reflect important aspects of network data, that affects how similarity measures behave.

In Table 1 is an overview of the criteria of the similarity measures we have chosen to focus on in the analysis.

	Connectivity	Local/Global	Discrete/Continuous	Attributed	Multi-relational
Jaccard	Yes	Local	None	None	Yes
Cosine	Yes	Local	None	None	Yes
MV	No	Global	Both	Yes	Yes
SPG	No	Local	Both	Yes	No
SPV	No	Global	Both	Yes	No
WL	No	Local	Discrete	No	No
node2vec	No	Local	Discrete	Yes*	Yes*

Table 1: Overview of method criteria

The criteria we consider can be split into two groups: *Applicability* and *Behavioural*. Discrete/Continuous, attributed and multi-relational falls under the *Applicability* group. These criteria represent the practical shortcomings of the different measures, for example, if the data you have only contain continuous attributes, it would not be possible to obtain useful results using WL.

The other group, *Behavioural*, which includes the connectivity and local/global criteria, represent how the similarity measures behave on the data. For some tasks, on certain networks which show a strong tendency of homophily, a measure such as Jaccard might be sufficient. However, for other networks, which contain many disconnected components, or where the attributes of the network carry important information, Jaccard's expressiveness may not be enough.

From Table 1 we see that the similarity measures are fairly varied in terms of what criteria they handle. Immediately we can see that both Jaccard and Cosine is not very applicable, as they both fully ignore any form of attributes. This means that these are not good measure in networks where using the attributes are important.

These are also the only measures requiring connectivity, as they produce the similarity score by looking at the immediate neighbours. If we look at the other measures, we see that WL is the least flexible, as it is not only restricted to discrete attribute values, it also cannot handle multiple attributes nor multi-relational networks. MV is the most flexible measure, as it works with both discrete and continuous attributes, as well as multi-attributed and multi-relational networks. The original node2vec implementation proposed by Grover et al., is less flexible than WL in terms of what network data it uses, as it ignores attributes entirely, using only vertex ids. Our proposed extensions make use of much more data available in networks.

## Part III

# Exploratory analysis

This part presents an exploratory analysis of node2vec. There are two primary goals of this analysis. The first goal is to examine the claims made by Grover et al.[6]. More specifically, we examine the claim that node2vec is able to find structural similarity with certain  $p$  and  $q$  parameters, and that changing these two parameters will cause node2vec to find proximity. The second goal of this analysis is to examine the behaviour of our node2vec extensions on attributed and multirelational graphs.

For this analysis we use a Python implementation of node2vec made available by Grover et al.<sup>2</sup>.

We start off with a short description of the procedure. Next we introduce the various datasets used throughout this analysis. After that we examine the claims made by the node2vec authors. Next we examine the behaviour of our attribute extension. Finally we have a case study on the airline network to examine the multirelational extension.

## 6 Procedure

This section covers some of the methods and metrics used for this analysis.

### 6.1 K-means

For this analysis we use k-means clustering on vertex embeddings to see how well node2vec distinguishes between different structural vertex types such as hub vertices.

K-means is a clustering algorithm where the goal is to find groups(i.e. clusters) in the data, where the number of groups is represented by  $k$ . K-means iteratively refines its result until it converges or a specified number of iterations has run.

The algorithm has three steps, which can be summarized as follows. First  $k$  centroids are randomly initialized. A centroid is a point in space which defines one of the  $k$  clusters.

Then each data point  $x$  is assigned to their closest centroid  $c_i$ , based on the squared euclidean distance. This is formally defined in Equation 38.

$$\arg \max_{c_i \in C} dist_{euc}(c_i, x)^2 \tag{38}$$

Where  $C$  is the set of centroids, and  $dist_{euc}(c_i, x)^2$  is the squared euclidean distance between centroid  $c_i$  and data point  $x$ .

Finally the position of each centroid is updated by taking the mean of all of its assigned data points, see Equation 39. Step two and three are repeated until it converges.

---

<sup>2</sup><https://github.com/aditya-grover/node2vec>

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i \quad (39)$$

Where  $S_i$  is the set of data points assigned to centroid  $c_i$ .

## 6.2 Adjusted rand index

We use the adjusted rand index to measure how well the clusterings generated by node2vec, compare to the manually determined structural role groups. A high value of the adjusted rand index means that the clustering is close to the ground truth. Consider two clusterings,  $X$  and  $Y$ , where  $X$  is the ground truth, then the adjusted Rand index is defined as in Equation 40.

$$RI_{adj} = \frac{2(ad - bc)}{(a + b)(b + d) + (a + c)(c + d)} \quad (40)$$

Here  $a$  are the number of pairs which are in the same cluster in  $X$  and in the same cluster in  $Y$ .  $b$  are the number of pairs which are in the same cluster in  $X$  but in different clusters in  $Y$ .  $c$  is the number of pairs which are in different clusters in  $X$  but in the same cluster in  $Y$ . Finally,  $d$  is the number of pairs which are in different clusters  $X$  and also in different clusters in  $Y$ .

## 6.3 Graph cluster cohesion

To quantify whether node2vec finds structure or proximity we measure the graph cluster cohesion, which is the average shortest path distance between any pair of vertices in a cluster. The assumption here is that if structural roles are discovered, the average distance will be high (low cohesion), while if proximity is discovered the average distance will be low (high cohesion). We formally define graph cluster cohesion in Equation 41.

$$cohesion(C_l) = \frac{-1}{|C_l| \cdot (|C_l| - 1)} \sum_{v, u \in C_l, v \neq u} dist_{sp}(u, v) \quad (41)$$

Where  $C_l$  is a cluster of vertices,  $|C_l|$  is the amount of vertices in  $C_l$ , and  $dist_{sp}(u, v)$  is the shortest path distance between vertices  $u$  and  $v$ .

## 7 Dataset overview

This section describes the datasets used in this analysis. A summary of the different datasets is shown in Table 2.

	V	E	Avg. Degree	Diameter	Multirelational
Airline	304	3076	19 (sd = 24)	5	Yes
Karate	34	78	4.59(sd = 3.89)	5	No
Les Miserables	77	254	6.6(sd = 6)	5	No

Table 2: Overview of the datasets used for the exploratory analysis

## 7.1 Airline

A dataset containing air routes were obtained from OpenFlights.org[2]. This dataset was found in two parts, the first part containing information about 3334 airports(vertices), the second part containing information about 67,663 flight routes(edges) operated by 547 different airlines(edge labels). We filter the data to include only airports situated in Europe.

OpenFlights provide their own internal id for the airports, which makes it possible to connect the airports to another dataset from OpenFlights. We use this id to connect the airport and the flight route datasets. Additionally, airports are also identified by their International Civil Aviation Organisation(ICAO) code, which we use to extract airport attributes from a third dataset.

The OpenFlight dataset does not contain any attributes by default, but an attribute we would like to include in the dataset, is the amount of passengers carried by each airport. A dataset containing passenger and goods information, from 1997-2016, is made available by the Eurostat project[1] for airports in EU member countries.

In the Eurostat dataset, airports are identified by country code and ICAO. For example, London Heathrow airport(ICAO:EGLL) will be denoted by UK\_EGLL. We remove the country code, which makes it possible to connect the Eurostat data to the OpenFlights data.

The OpenFlights and Eurostat data both contain some airports that are not in the other dataset. We keep only the airports that exist in both. This leaves us with 304 airports, all of which have a single attribute denoting the amount of passengers carried.

We divided the airports into three different categories based on their passenger amount, representing small, medium and large airports, see Table 3.

category	small	medium	large
interval	]2248, 496 000]	]496 000, 10 000 000]	]10 000 000, 75 000 000]
nr of airports	102	162	40

Table 3: The intervals

An example of the route network for the Lufthansa airline can be seen in Figure 37

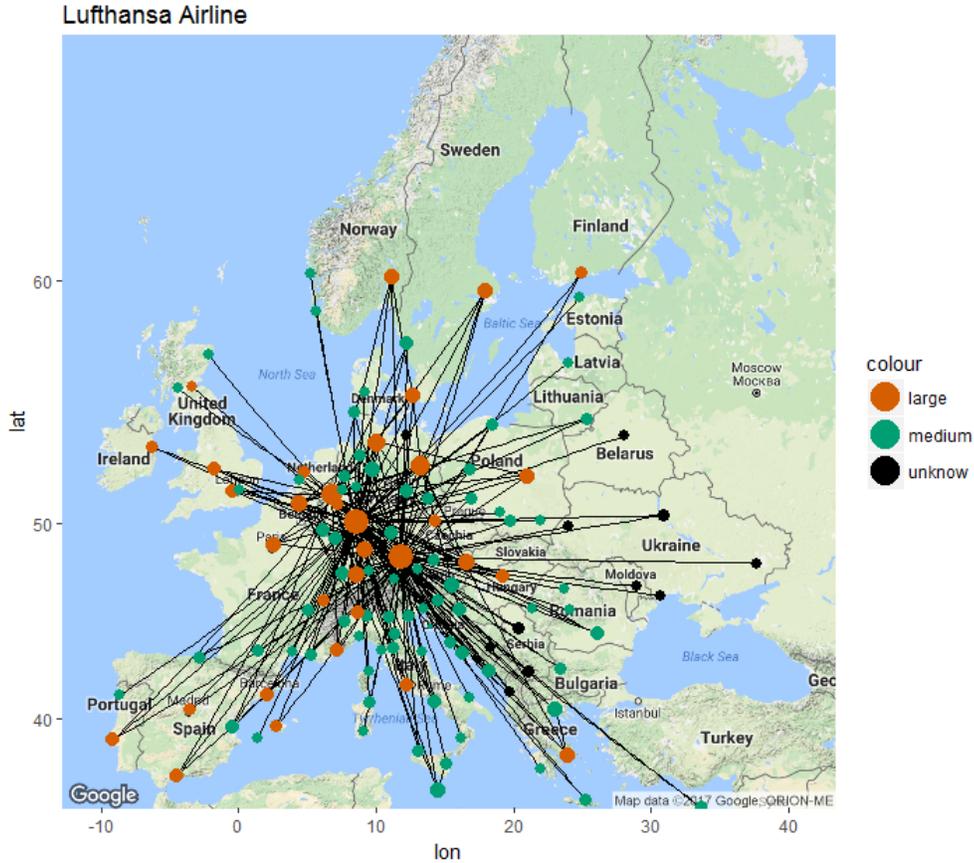


Figure 37: Lufthansa airline

For our exploratory analysis we consider a multirelational version of this airline data, where there is an edge between two airports if one of the ten largest airlines has a route between the airports. We determine what are the largest airports by the amount of flights. The top ten airlines can be seen in Table 4. In summary, this leaves us with a dataset with 304 airports (vertices), 3076 flight routes (edges), and 10 airlines (relation types).

Airline	Nr of routes
Ryanair	1122
easyJet	460
Air Berlin(AB)	323
Iberia Airlines(IB)	285
Norwegian air shuttle	260
Lufthansa(LH)	250
Alitalia(AL)	239
Germanwings(GW)	218
SAS	214
Wizz air(Wizz)	209

Table 4: Top ten largest airlines

## 7.2 Zhachary Karate Club

The Zhachary Karate Club dataset, referred to as Karate, is a social network modeling the members of a university karate club[20]. It can be represented as a undirected unweighted graph, which contains 34 vertices(members of the karate club), one member being an instructor, and one being an administrator. Furthermore is has 78 edges, representing interaction between members outside club activities.

## 7.3 Les Misérables

The Les Misérables dataset is a network modelling characters and coappearances of characters in the novel "Les Misérables" by Victor Hugo[9]. It can be represented as an undirected graph with 77 vertices and 254 edges. Vertices are characters and edges are coappearances. Edges exists between two characters if they appear in the same chapter of the book.

# 8 Node2vec unattributed graph experiments

This section examines the behaviour of node2vec without attributes. The primary goal of this section is to verify the claim by the authors, that node2vec is able to find structural similarity and proximity, and that you can adjust which type it finds by changing the  $p$  and  $q$  parameters. To do this, we test node2vec on the Zachary Karate Club network. First we find a set of parameters where node2vec succeeds in finding structural similarity. Next, we examine the effect of changing the parameters of node2vec. Finally we use what we learned from the karate network to examine the Les Misérables case study from their article.

We use the euclidean distance between vertex embeddings to measure how similar node2vec finds two vertices. Additionally, we cluster the vertices into clusters based on their embeddings using k-means. We use three clusters that should ideally represent hub vertices, periphery vertices, and mainstream vertices.

We manually determined whether each vertex were a hub, periphery or mainstream vertex, see Table 5.

	Vertices
Hubs	0, 1, 2, 32, 33
Mainstream	3, 5, 6, 7, 8, 10, 13, 19, 23, 27, 28, 29, 30, 31
Periphery	4, 9, 11, 12, 14, 15, 16, 17, 18, 20, 21, 22, 24, 25, 26

Table 5: The structural roles of vertices in the Zachary network

## 8.1 Structural parameters

To verify that node2vec can find structural similarity, we use the same  $p$  and  $q$  parameters as the authors. The other parameters were not specified, so we found them

through trial and error. The set of parameters where node2vec is able to find structural similarity on the karate network are:

$$dim = 8, wl = 5, numWalks = 100, windowSize = 3, p = 1, q = 2$$

In this section we show the results of node2vec with these parameters.

First we want to see whether node2vec can find hub vertices. In Figure 38, hub vertex 33 is compared to the other vertices. We expect it to be similar to other hub vertices. The vertices are coloured on a gradient from red to blue, representing the distance of their embeddings to the embedding of vertex 33. Red is minimum distance and blue is maximum distance.

The vertices that are closest to 33 are vertices 0, 1, 2, and 32. These are all of the hub-vertices, so node2vec is able to find those.

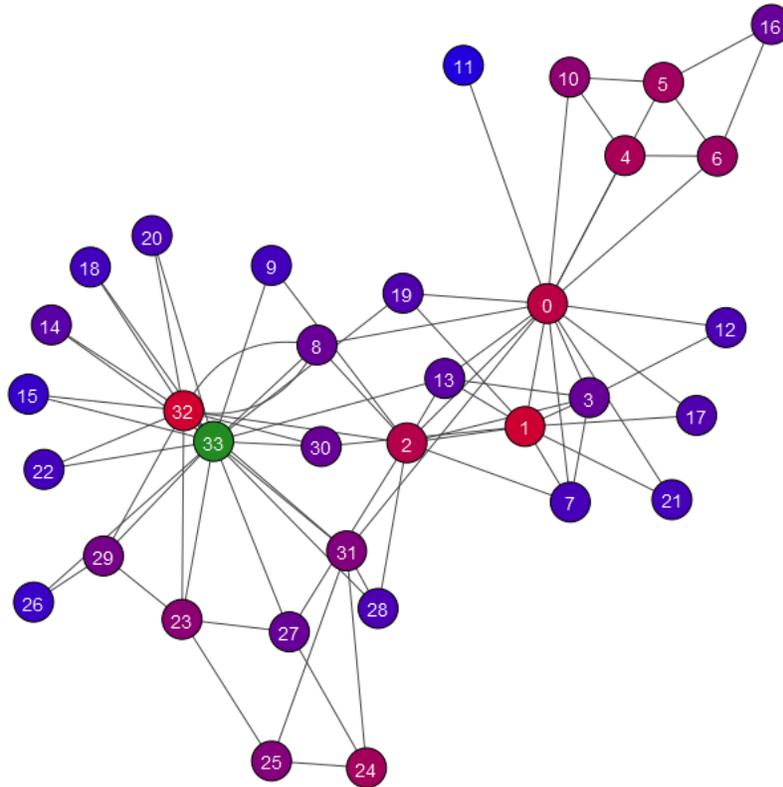


Figure 38: Distance between hub vertex 33 and other vertices

Figure 39 shows the same comparison of vertices to the periphery vertex 11. This shows that the vertices closest to vertex 11 are periphery vertices, while the hub-vertices have the highest distance to vertex 11. This means that node2vec does distinguish between hub vertices and periphery vertices.

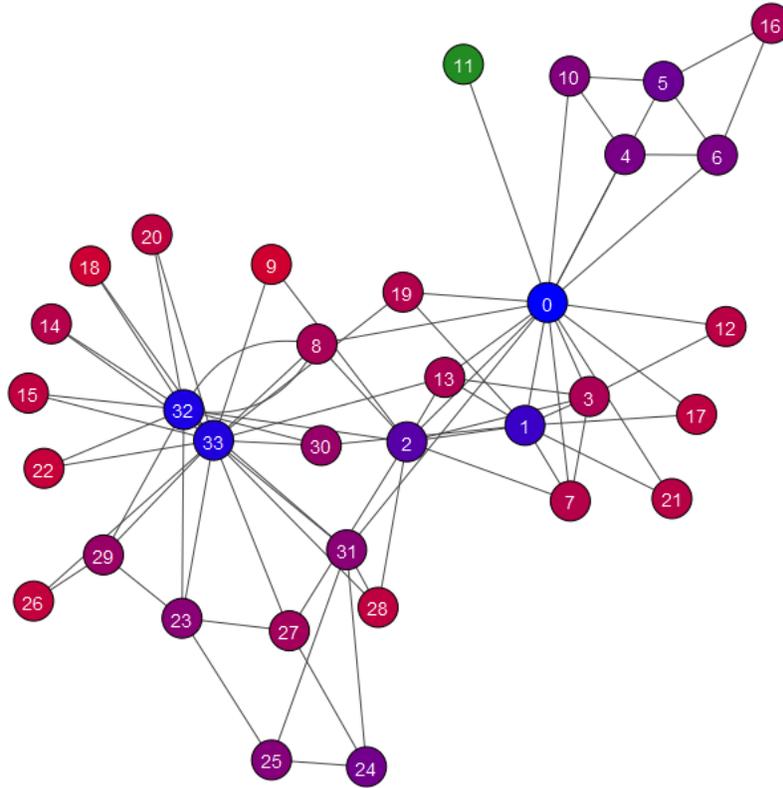


Figure 39: Distance between vertex 11 and other vertices

To test whether `node2vec` can distinguish all three structural roles we look at the clusterings. In Figure 40 we see the result of this. All of the hub vertices except vertex 2 are in the red cluster, and all of the periphery vertices except vertices 4, 24 and 25 are in the green cluster. The mainstream vertices are spread between the blue and green clusters. From this, it appears that `node2vec` can detect hub vertices and periphery vertices, but it clusters some mainstream vertices with the periphery vertices.

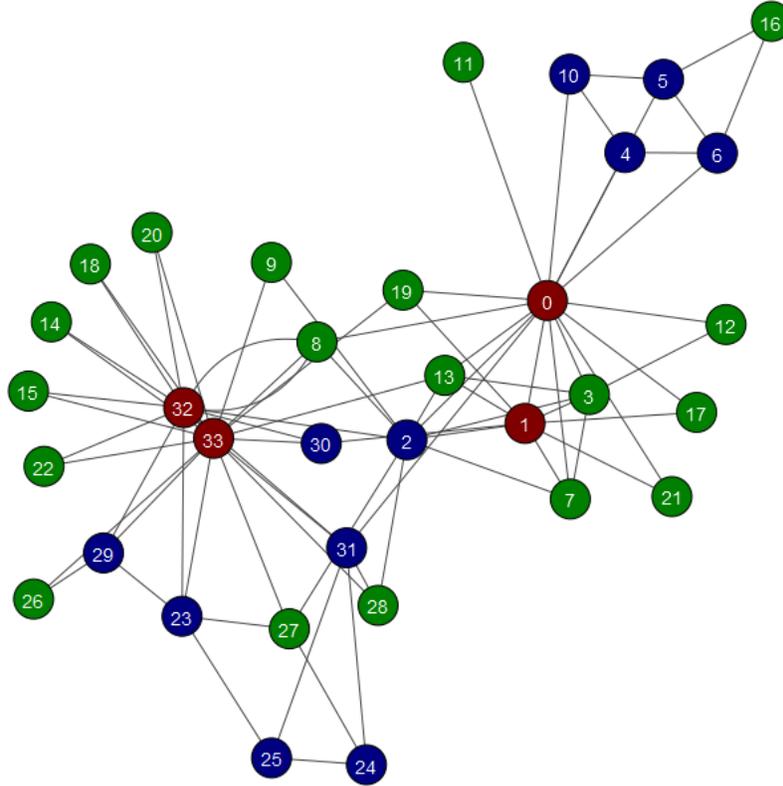


Figure 40: k-means clustering on karate with 3 clusters

To examine how well node2vec distinguishes these three structural roles, we used multidimensional scaling (mds) to get a visual representation of the embeddings. The points are coloured according to their structural role, where red is hub vertices, green is periphery vertices, and blue is mainstream vertices. In Figure 41, we see a clear separation between the hub vertices and the other two roles. The separation between periphery vertices and mainstream vertices is less clear. There is a bit of separation at the top right corner, but the rest of them are fairly mixed. This is consistent with the graph structure, where the difference in degree between the hub vertices and mainstream vertices, is bigger than the difference between the mainstream vertices and the periphery vertices. From this, it appears that node2vec correctly finds structural similarity in this network, but struggles a bit to distinguish mainstream vertices from periphery vertices.

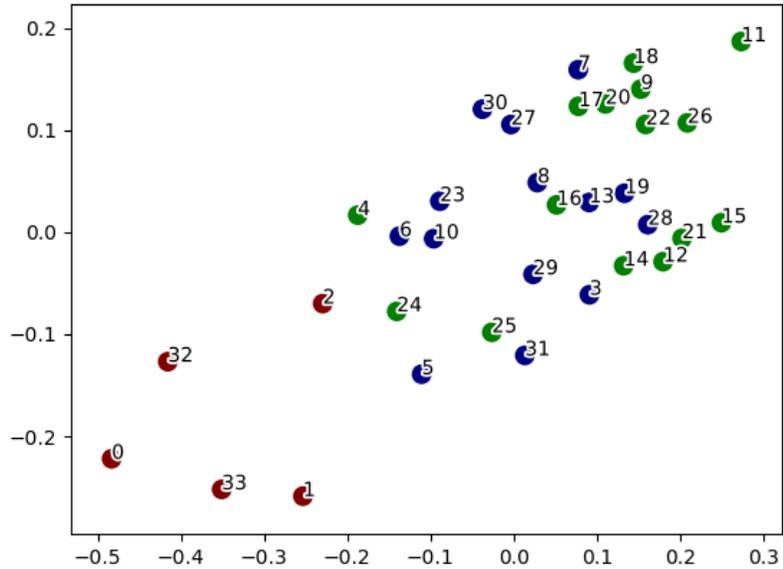


Figure 41: Multidimensional scaling of the embeddings given by node2vec

Finally we wanted to test whether node2vec works on disconnected graphs. To do this, we created the duplicated karate graph. Node2vec could not find structural similarity on this graph with the same parameters as above. Each vertex was more similar to all other vertices in the same disconnected subgraph, than to any vertex in the other subgraph. If we lower the walk length from five to four, node2vec is able to find structural similarity like in the original graph. In Figure 42 we see the clustering with walk length four. The results are similar to the clustering on the karate network.

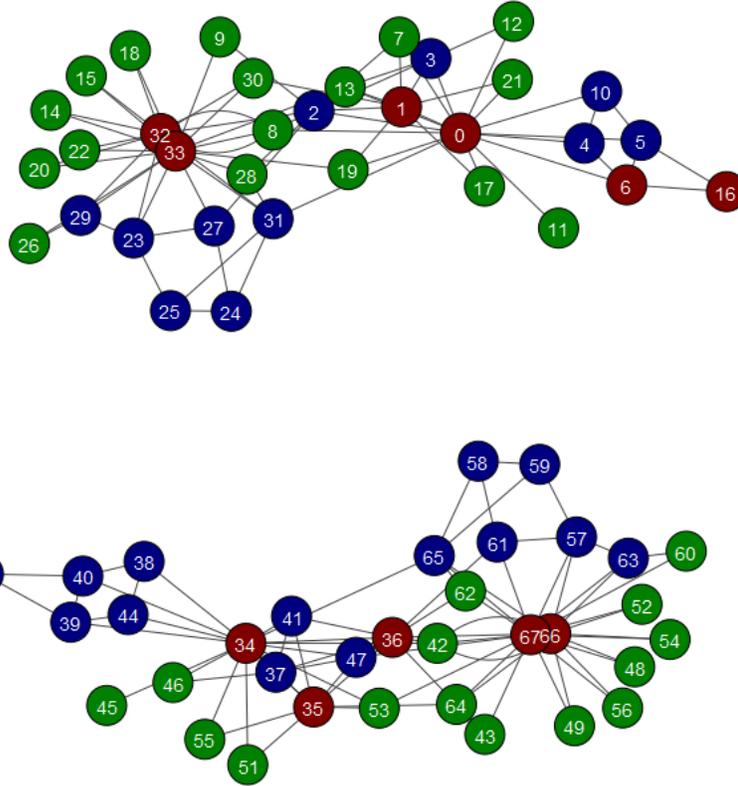


Figure 42: k-means clustering on duplicated karate with 3 clusters

## 8.2 Changing parameters

The following subsections examine the effect of changing the parameters of node2vec. We use the structural parameters from subsection 8.1 except for the one parameter we change.

The main claim of the authors is that changing the  $p$  and  $q$  parameters, allows node2vec to discover either structure or proximity. To quantify how well node2vec discovers structure we use the adjusted Rand index. In addition, we use graph cluster cohesion to quantify if node2vec finds proximity.

### 8.2.1 Dimensions

We first tested the effects of changing the amount of dimensions. We tested from 1 to 1024 dimensions. When manually inspecting the clusterings there did not appear to be any difference between the dimension amounts.

To investigate the differences more closely we look at the cohesion and adjusted Rand index as a function of the amount of dimensions, as seen in Figure 43. Node2vec was run 100 times for every dimension amount. The x-axis shows the amount of dimensions, the y-axis shows the mean cohesion over the 100 runs, and the error bars show the standard deviation.

For dimensions of one and two the cohesion and rand index is a little bit lower

than at higher amount of dimensions. This suggests that the results are a bit more random at one and two dimensions. At dimensions four and above the results are relatively stable with very little difference between them. The standard deviation for cohesion is high at dimension one which suggests that the results are more random here, while the standard deviation is similar for all other dimensions.

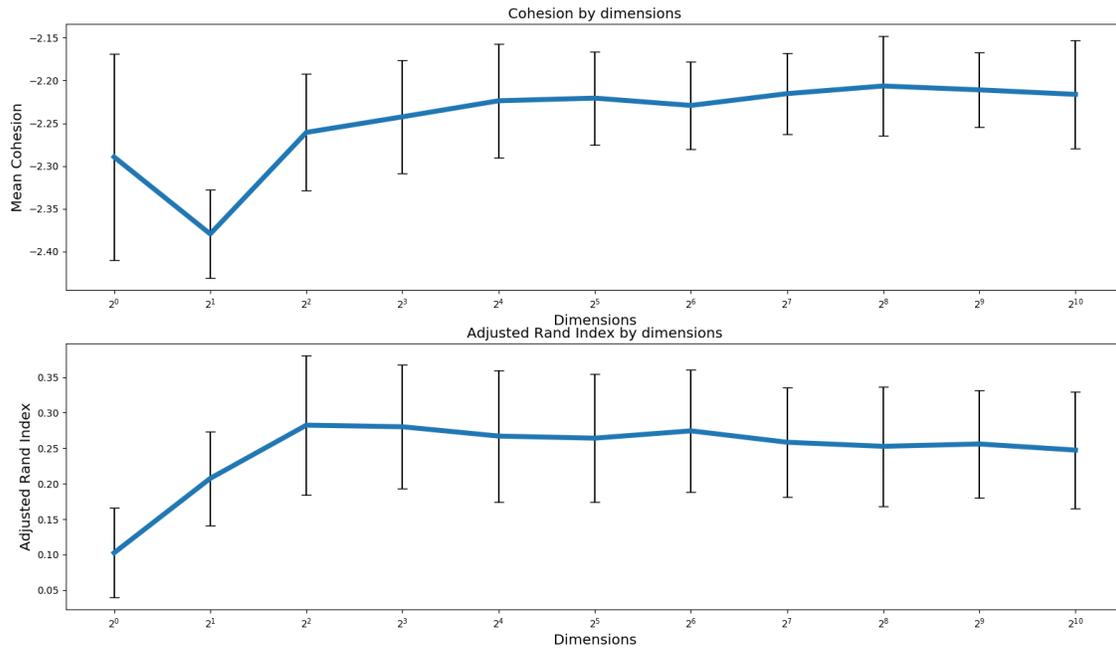


Figure 43: Cluster cohesion and adjusted Rand index for 1 to 1024 dimensions

### 8.2.2 Walk length

We also tested the effect of changing the walk length. We tested for walk lengths 1-20. In Figure 44 we see how the walk length affects the cohesion and adjusted Rand index.

For walk lengths 1-3 the cohesion and adjusted Rand index are both low, which indicates that for these walk lengths neither proximity nor structure can be found.

At walk lengths of 4-5 there is a noticeable jump in the adjusted Rand index, and a slight increase in cohesion. This suggests that node2vec is able to find structural similarity at these values. This is consistent with the results of the previous section. At these walk lengths the standard deviation is significantly larger than at other walk lengths, which suggests that there is some randomness in the results.

At a walk length of six node2vec starts showing a tendency towards proximity as the cohesion starts rising while the adjusted Rand index drops rapidly. At a walk length of seven and above node2vec finds proximity as both the cohesion and adjusted Rand index more or less remain stable at -1.6 and zero respectively.

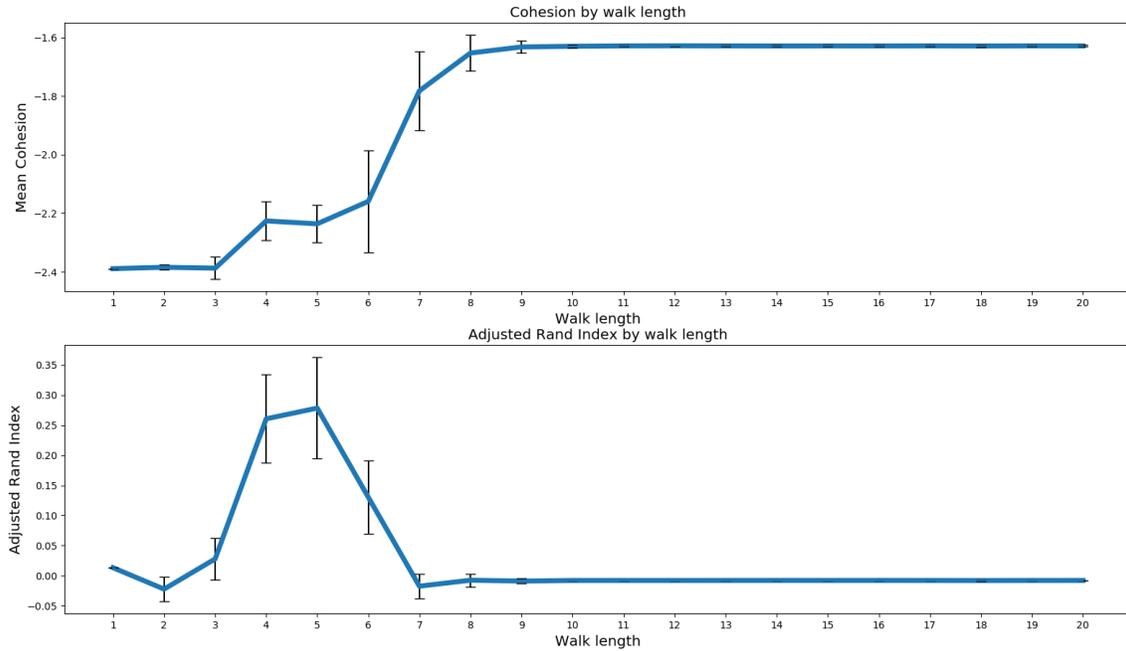


Figure 44: Cluster cohesion and adjusted Rand index for walk lengths 1-20

To confirm the findings given by the cohesion and adjusted Rand index, we look at the clusterings. In Figure 45 we see the clustering for walk lengths four and seven.

As we can see, with a walk length of four all the hub vertices are clustered in the red cluster, while most of the periphery vertices are in the green cluster. The mainstream vertices are spread between the green and blue clusters. Overall, these results are quite similar to those for the structural parameters.

For walk length seven all vertices are clustered together with their neighbour vertices. It appears, that node2vec cannot find structural similarity with this walk length, but finds proximity instead.

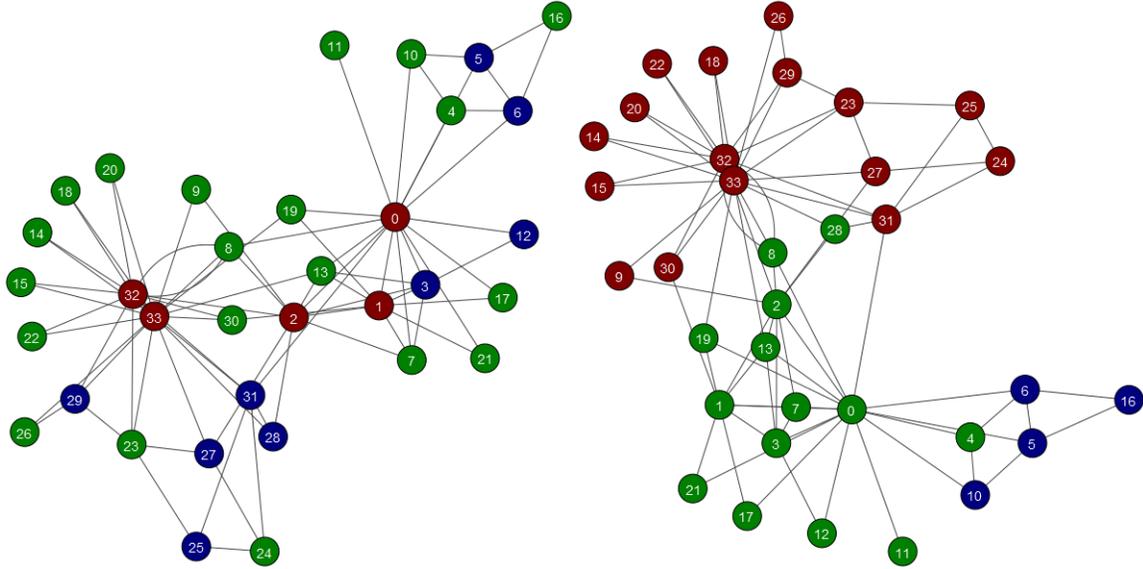


Figure 45: k-means clustering on karate with three clusters and walk length four (Left), walk length seven (Right)

To look closer at how node2vec distinguishes the clusters, we look at the mds representations for these walk lengths, seen in Figure 46. In these figures the points are coloured based on their structural role, where red points are hub vertices, blue points are mainstream vertices and green points are periphery vertices.

For walk length four we see a clear separation between the hub vertices and non-hub vertices. There is some separation between the periphery vertices and mainstream vertices as well, but there is a bit of overlap between the two roles.

For walk length seven there is no clear separation between the three roles. Some hub vertices are in the bottom left and some are in the top right. Vertices are embedded close to their neighbours in the graph, instead of being close to other vertices of the same role.

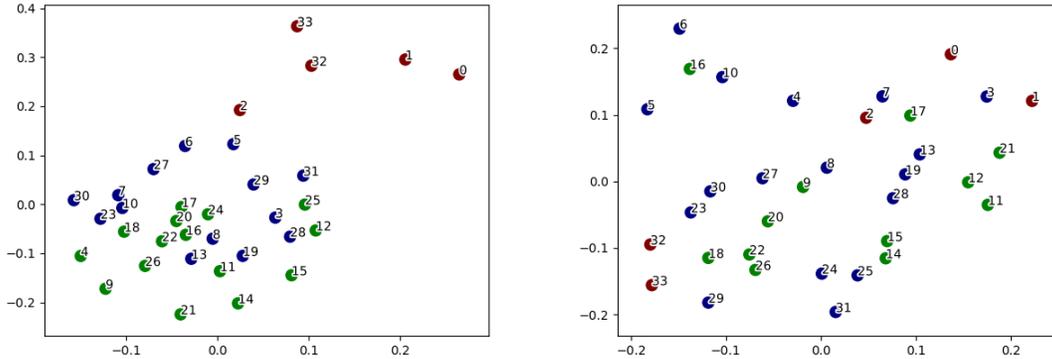


Figure 46: Multidimensional scaling of the embeddings given by node2vec with a walk length of four (Left), and walk length seven (Right)

### 8.2.3 Number of walks

For the number of walks we tested from 10 to 250 by increments of ten. We show how the graph cohesion and adjusted Rand index is affected by the number of walks in Figure 47.

The figures are very similar to the figures for walk length. At low amount of walks it is unable to find any structure at all, like with short walk lengths. Somewhere in the middle it is able to find structure, and at large amount of walks it finds proximity like with long walk lengths. Overall, changing the walk length and number of walks have the same effect on the results.

For cohesion, the standard deviation appear to increase as number of walks increases, while the adjusted Rand index has the largest standard deviation in the middle.

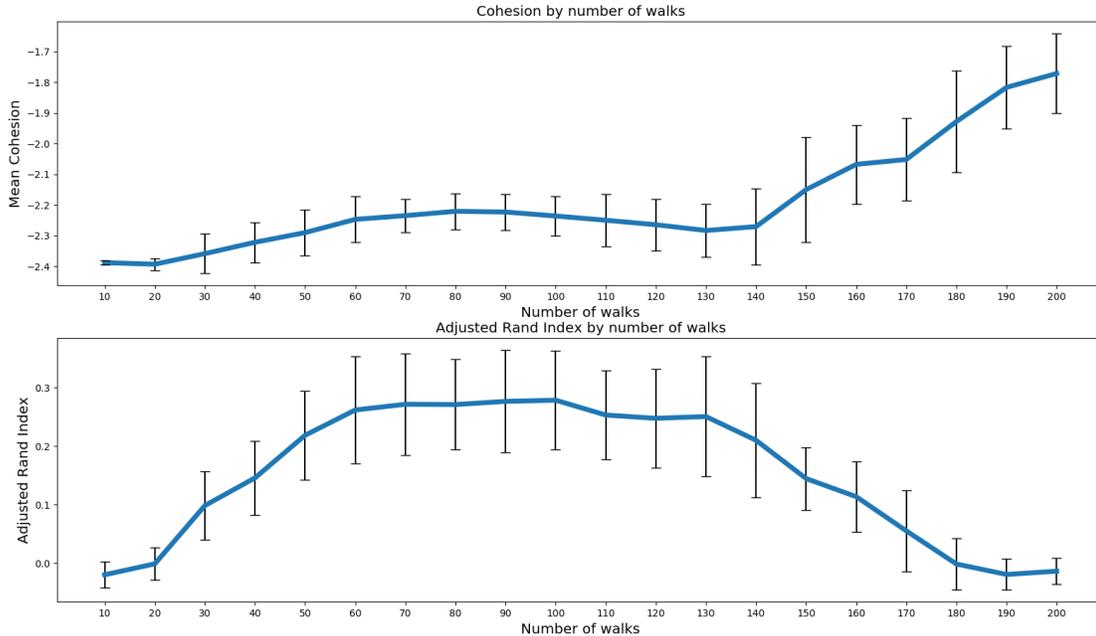


Figure 47: Cluster cohesion and adjusted rand index for 10-250 walks

### 8.2.4 Walk length and Number of walks

It seems that increasing the walk length has a similar effect to increasing the number of walks. This is most likely because both parameters control how much data is used for training. The amount of data  $D$  can be explained by Equation 42.

$$D = n \cdot \text{walk length} \cdot \text{number of walks} \quad (42)$$

Where  $n$  is the number of vertices.

In Figure 48 and Figure 49 we can see a heat map of cohesion and adjusted Rand index respectively, as a function of walk length and number of walks. From the cohesion heatmap, we see that increasing either the walk length or number of walks increases the cohesion, which leads to proximity. However, it needs a walk length of at least five before it finds proximity.

From the adjusted Rand index heatmap, we see that there are certain combinations of walk lengths and number of walks, where the clusterings are closer to the ground truth. Increasing or decreasing either of the parameters, causes the clusterings to get worse. This suggests that there is a sweet spot for the amount of data that allows node2vec to find structural similarity.

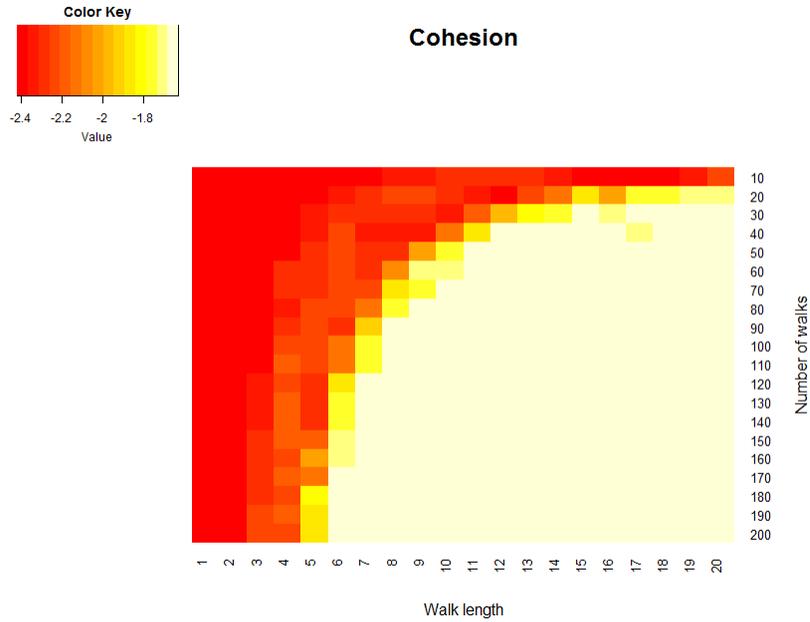


Figure 48: Heat map of cohesion as a function of walk length and number of walks

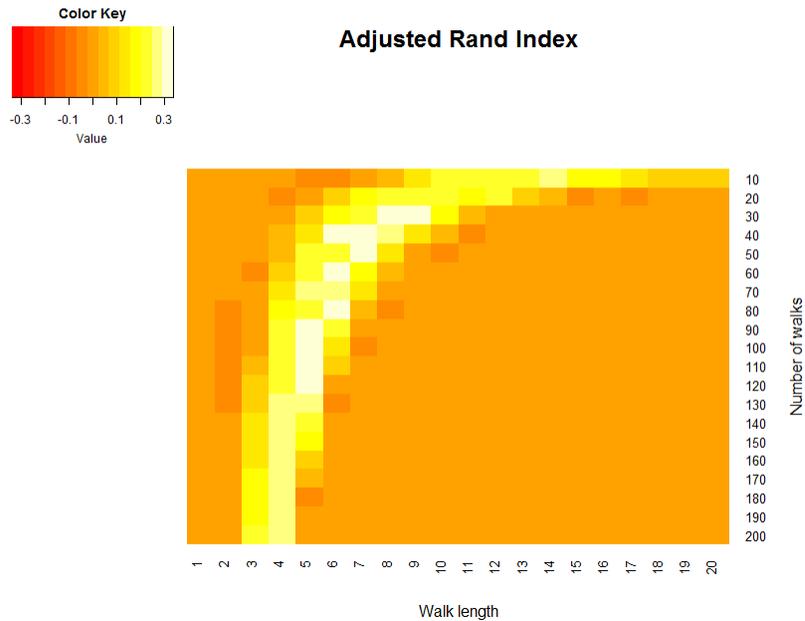


Figure 49: Heat map of adjusted rand index as a function of walk length and number of walks

### 8.2.5 $p$ and $q$

Finally we want to examine how `node2vec` behaves as we change the  $p$  and  $q$  parameters. These are supposed to be the main parameters for deciding which type of

similarity node2vec should find, i.e. structure or proximity. We test on  $p$  and  $q$  values of 0.2 to 5.

First, we test using the structural parameters from subsection 8.1. Looking at the clusterings for different combinations of  $p$  and  $q$  show little difference between them. To determine the effect of  $p$  and  $q$  on the results, we look instead at the cohesion and adjusted Rand index. In Figure 50 and Figure 51 we see a heat map of the cohesion and adjusted Rand index as a function of  $p$  and  $q$ . The cohesion does not seem to follow any pattern, and it is roughly the same for all combinations of  $p$  and  $q$ . This suggests that changing  $p$  and  $q$  does not have any effect on whether node2vec finds communities.

The adjusted Rand index shows something different though. It seems that low values of  $q$  make clusters that are closer to the ground truth than high values of  $q$ . Changing the  $p$  value does not have as large an effect, but it seems to generate slightly better clusters at high  $p$  values. According to the authors, low  $q$  values and high  $p$  values correspond with a depth first search, which should find communities instead of structure. However, our results seem to indicate that depth first search is better at finding structure.

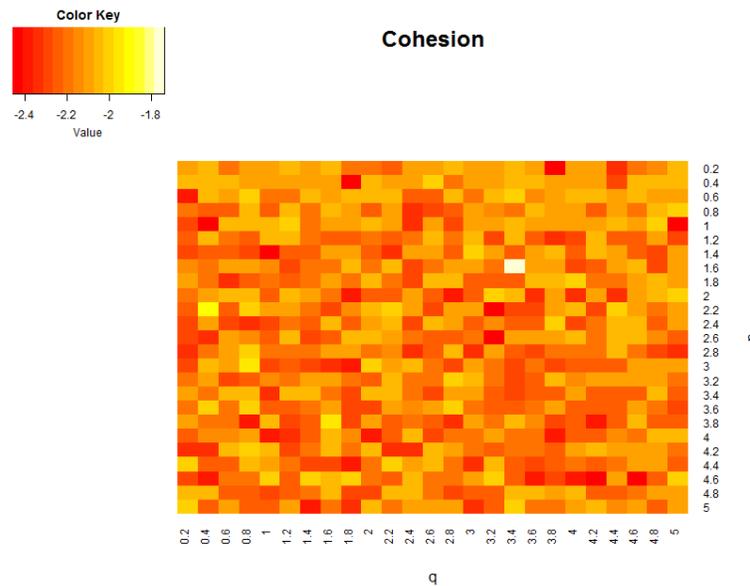


Figure 50: Heat map of cohesion as a function of  $p$  and  $q$  with structural parameters

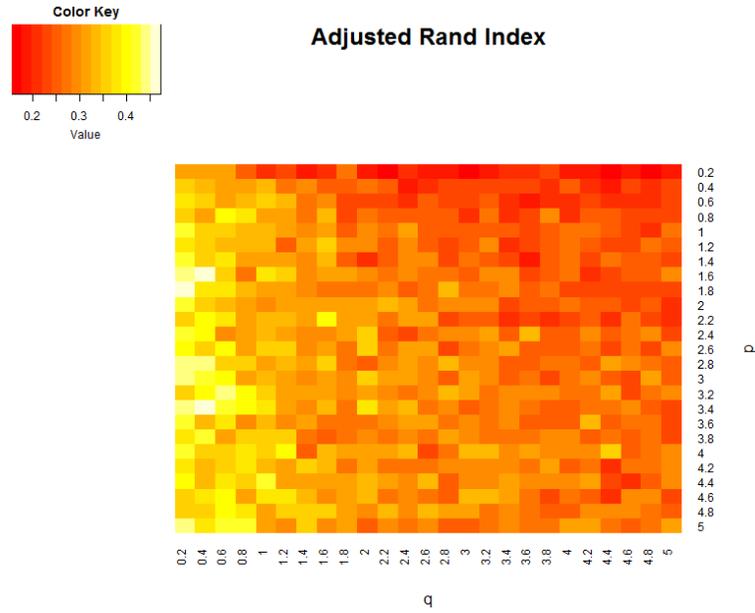


Figure 51: Heat map of adjusted rand index as a function of  $p$  and  $q$  with structural parameters

Next we want to see if changing  $p$  and  $q$  has any effect with parameters where node2vec finds proximity. In Figure 52 and Figure 53 we see the heatmaps with the walk length increased to ten. The cohesion and rand index does not change in any noticeable way when  $p$  and  $q$  are changed. This suggests that when node2vec finds proximity, changing  $p$  and  $q$  will not make it find structure instead.

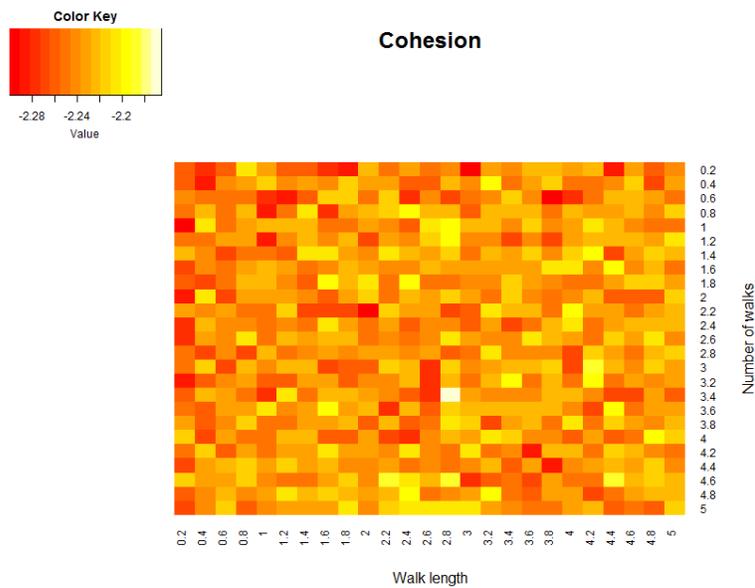


Figure 52: Heat map of cohesion as a function of  $p$  and  $q$  with proximity parameters

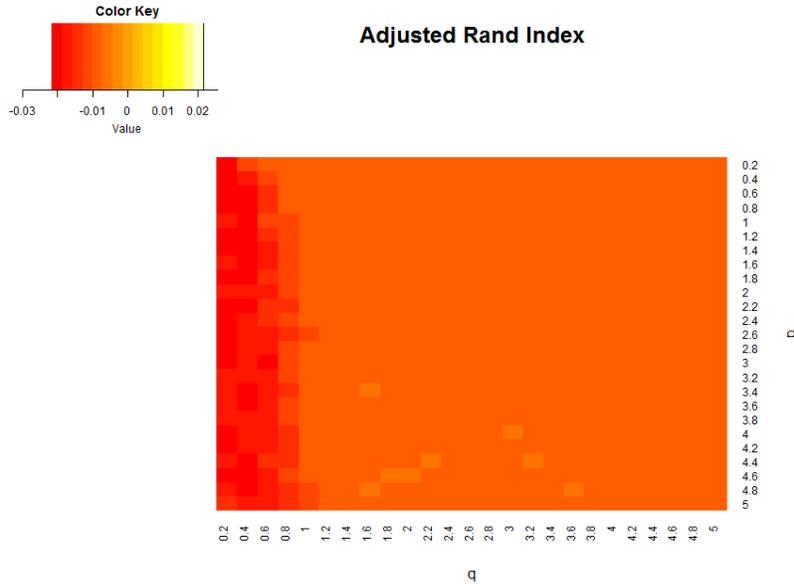


Figure 53: Heat map of adjusted rand index as a function of  $p$  and  $q$  with proximity parameters

### 8.3 Les Misérables

In this section, we use what we learned from the karate network to further examine the authors claims. In the article they do a case study on the Les Misérables network, where they show the results with 16 dimensions and two different combinations of  $p$  and  $q$ . They do not say what other parameters they use. They find that node2vec is able to find structural similarity with  $p = 1$  and  $q = 2$ , and proximity with  $p = 1$  and  $q = 0.5$ .

To verify their claims we test node2vec with  $p = 1$  and  $q = 2$  on the Les Misérables network, and vary the walk length and amount of walks until we find structural similarity. The parameters that find structural similarity are:

$$dim = 16, wl = 3, numWalks = 100, windowSize = 3, p = 1, q = 2$$

In the left plot in Figure 54 we see the clusters using these parameters, while the right plot shows the clustering with walk length five.

With walk length three node2vec finds a bit of structural similarity. All of the green vertices are hub vertices, and the red ones tend to be periphery and mainstream vertices. The blue cluster seems to be a mix of all three structural roles. With a walk length of five node2vec finds proximity. This seems to mirror the results of the karate network, where increasing the walk length makes node2vec find proximity.

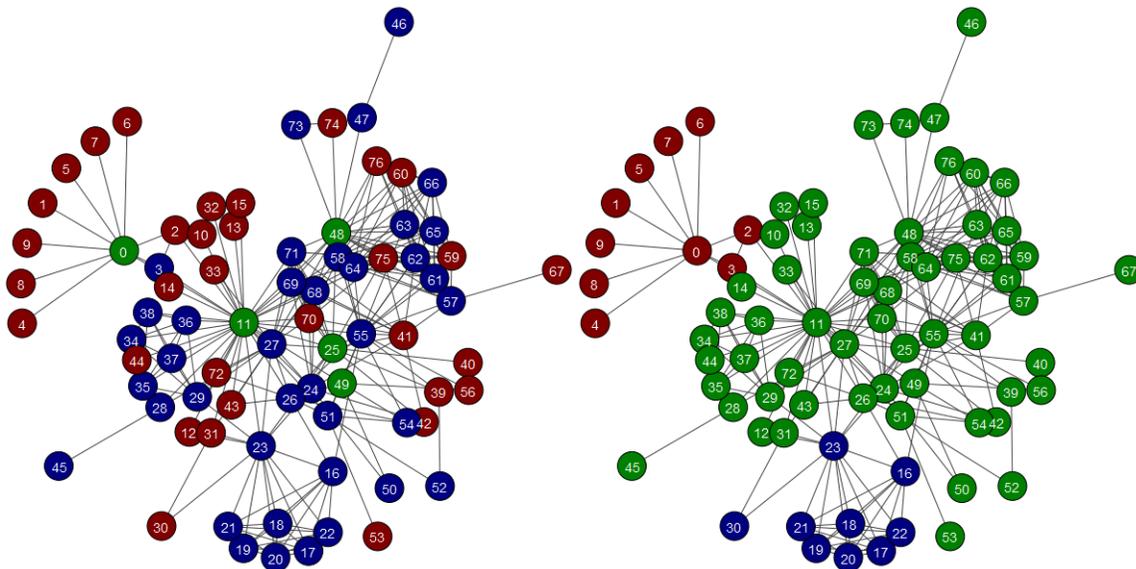


Figure 54: (k-means clustering on Les Misérables with 3 clusters and walk length 3 (Left), walk length 5 (Right))

Since we found some parameters that finds structural similarity on the Les Misérables network, we want to examine whether changing the  $p$  or  $q$  values can cause node2vec to find proximity. To test this, we measure the cluster cohesion of different  $p$  and  $q$  values to see if changing them makes a difference. In Figure 55 we see the cohesion as a function of  $p$  and  $q$ . As we can see changing  $p$  or  $q$  does not change the cohesion noticeably, which suggests that changing  $p$  and  $q$  does not cause node2vec to find proximity.

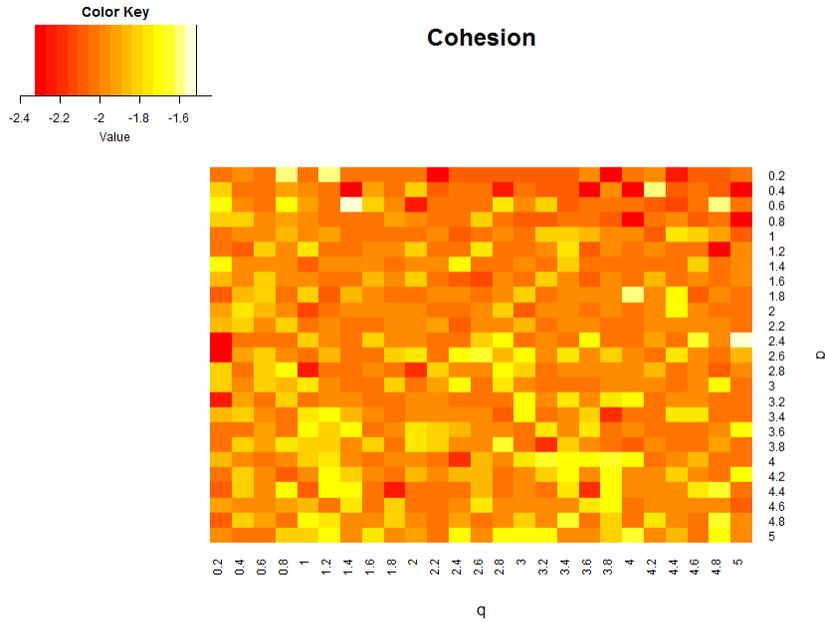


Figure 55: Heat map of cohesion as a function of  $p$  and  $q$  when node2vec finds structure

Next we look at the case where the authors claim that node2vec finds proximity, with  $p = 1$  and  $q = 0.5$ . We find the same results as above. Parameters (dim: 16, walkLength: 5, numWalks: 100, window: 3, iter: 2, p: 1, q: 0.5) finds proximity, and changing walk length to three finds structure.

Keeping the parameters that find proximity, we want to determine if changing  $p$  or  $q$  can make node2vec find structure. In Figure 56, we see a heatmap of the cohesion as a function of  $p$  and  $q$ . As we can see, changing  $p$  and  $q$  does not change the cohesion, which once again shows that changing  $p$  and  $q$  does not cause node2vec to find structure. This contradicts the authors claims that changing  $p$  and  $q$  changes whether node2vec finds structure or proximity.

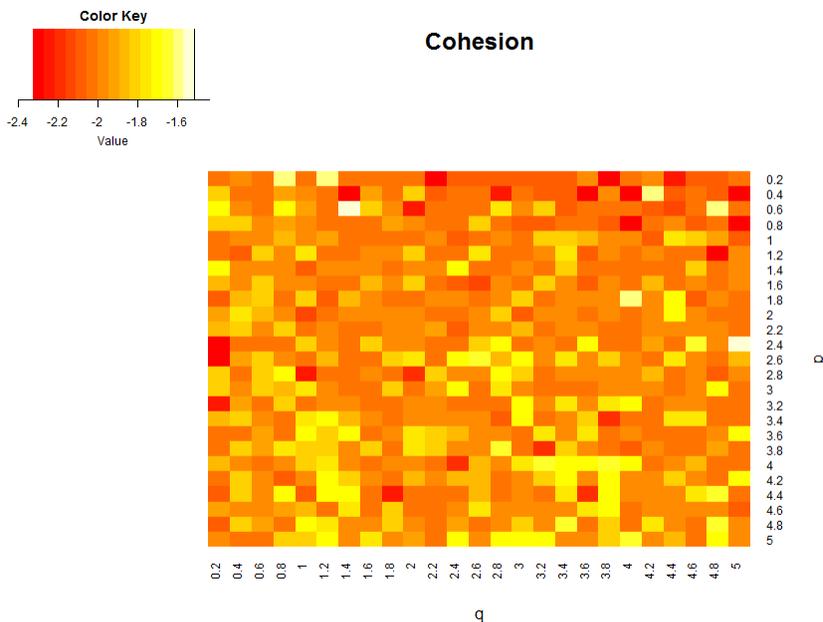


Figure 56: Heat map of cohesion as a function of  $p$  and  $q$  when node2vec finds proximity

## 9 Node2vec attributed experiments

In this section we investigate effect of adding attributes to node2vec using our extension node2vec-attr. In particular, we want to see if it successfully combines structural and semantic similarity. We do this by examining the clustering and mds results of node2vec-attr, and compare these to the results for node2vec. As a reminder, in the walks of node2vec-attr, vertices are represented as follows:

$$\phi_{attr}(v) = id, attribute$$

The experiments for this are done on the synthetic network *5Comm*, which is described further in subsection 9.1. The experiments and results are shown later in subsection 9.2.

### 9.1 5Comm

To test our extension node2vec-attr we constructed a synthetic graph referred to as *5Comm*, which can be seen in Figure 57. The graph has 29 vertices and 28 edges. In order to include attributes, we assign each vertex of *5Comm* one of four colours  $\{blue, red, black, green\}$ . These vertex colours are assigned in such a way that a local community is formed for each colour, resulting in network being split into five communities thus the name *5Comm*. The colour assignments for the vertices of the network are shown in the table below.

Colour	Vertices
Red	0-5
Blue	6-10 and 18-22
Green	11-17
Black	23-28

Table 6: Assignment of vertex colours in the 5Comm network

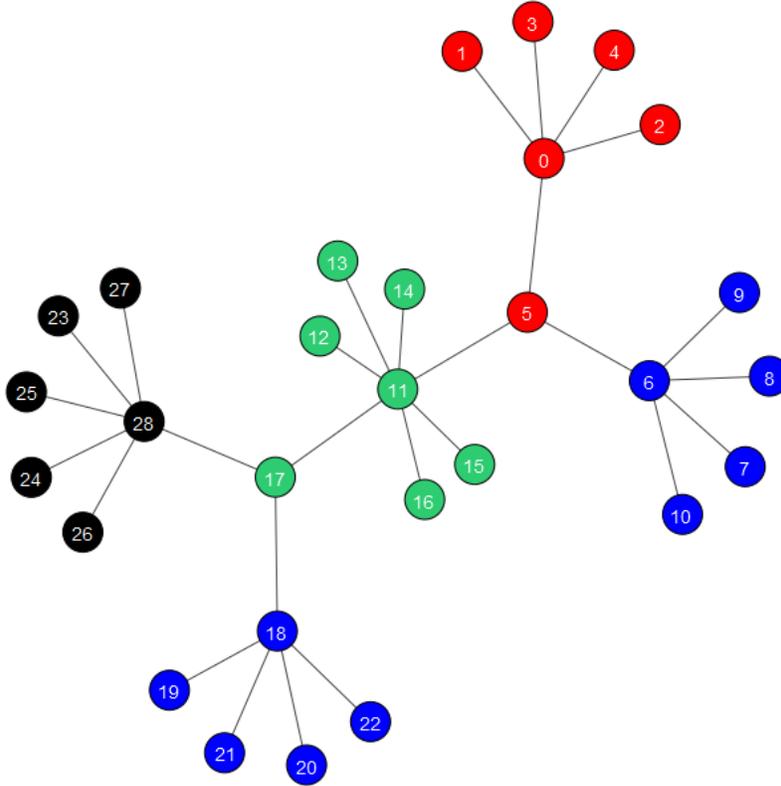


Figure 57: 5Comm with two blue communities

As in section 8 we consider three structural roles, namely, hubs, periphery and mainstream vertices. We manually defined them for this network, as shown in Table 7.

	Vertices
Hubs	0, 6, 18, 28, 11
Mainstream	5, 17
Periphery	1, 2, 3, 4, 10, 9, 7, 8, 13, 16, 15, 14, 12, 19, 21, 22, 20, 23, 24, 25, 26, 27

Table 7: The structural roles of vertices in the 5Comm network

## 9.2 Experiments

For the experiments on 5Comm we chose the following parameters, identical to the ones used for the karate network, as we found that they were also fit for finding

structure in the 5Comm network:

$$dim = 8, wl = 5, numWalks = 100, windowSize = 3, p = 1, q = 2$$

To determine what types of similarity node2vec-attr can find, we investigate whether the clusterings correspond to the communities of Figure 57.

For the purpose of comparison, we first conduct experiments for node2vec. Figure 58 shows the result of node2vec for k-means clustering with  $k = 4$ , with the colours representing the clusters. The corresponding mds plot is seen in Figure 59, where the points are coloured according to their real community.

From this, we see that the hubs; 0, 6, 11, 18 and 28 are clustered together, while the vertices of other structural roles are mixed. This is also visible in the mds where the hubs are clearly separated from the rest of the vertices, while mainstream vertices 5 and 17 are mixed along with the periphery vertices. In other words, some notion of structure is found.

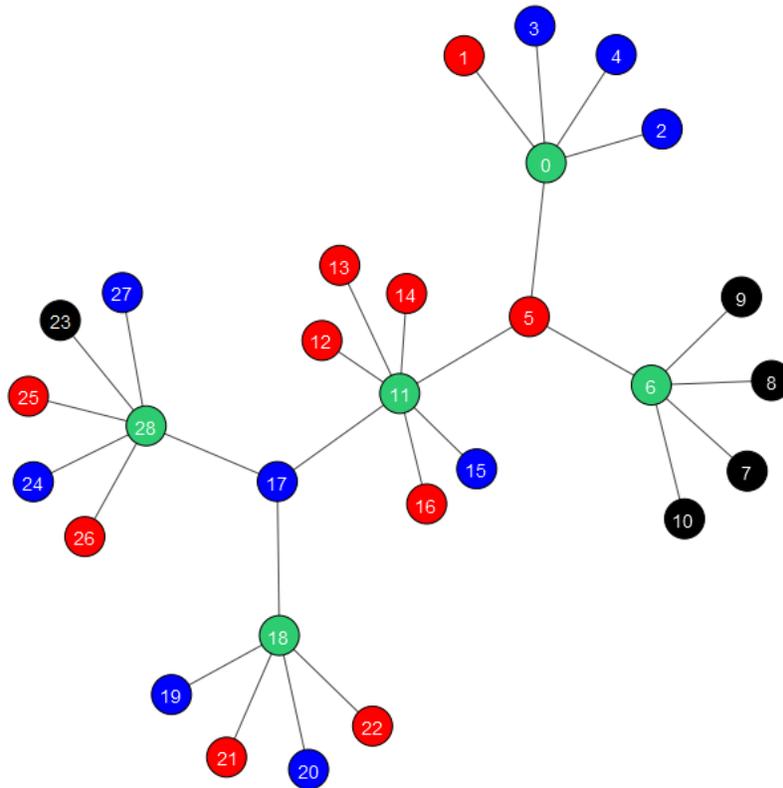


Figure 58: 5Comm unattributed for kmeans with  $k = 4$

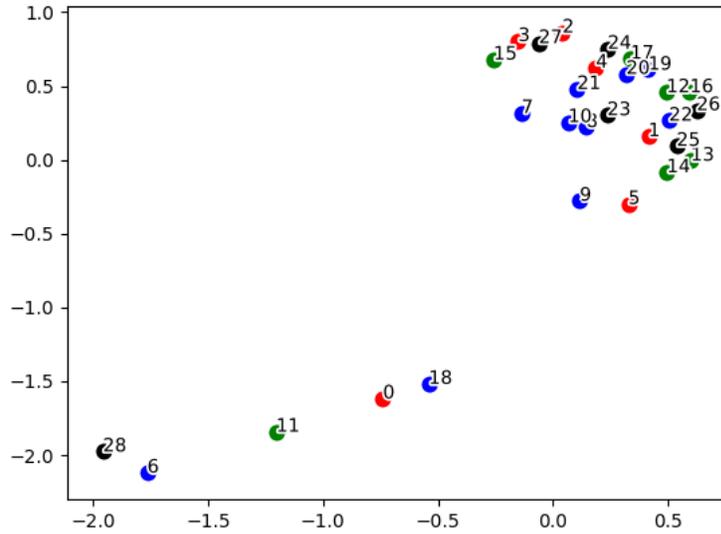


Figure 59: 5Comm unattributed mds for k-means with  $k = 4$

Figure 60 shows the result for node2vec-attr when clustering with k-means for  $k = 4$ , and the corresponding mds plot in Figure 61. For these experiments the window size is doubled to windows size = 6 in order to account for the inclusion of the attributes.

Here the resulting clustering is the same as the communities of the original 5Comm network. The vertices of the two blue communities are clustered together, which suggests semantic similarity. However, in the mds there is a slight separation between the two blue communities, which means that the results are likely to be somewhat proximity based. Furthermore, all vertices have a short distance to their colour embedding, except the two mainstream vertices 5 and 17. These two vertices are somewhat separated in the middle, but they are still closer to their immediate neighbourhood communities, as well as the color embedding they belong to. For example, vertex 17 is closer to the immediate neighbourhood communities of hubs 11(green), 18(blue) and 28(black), than to the other communities. Additionally, it is also closer to the green colour embedding, and community, than to any other communities. Thus, the clustering here is based on a mix of semantic similarity and proximity.

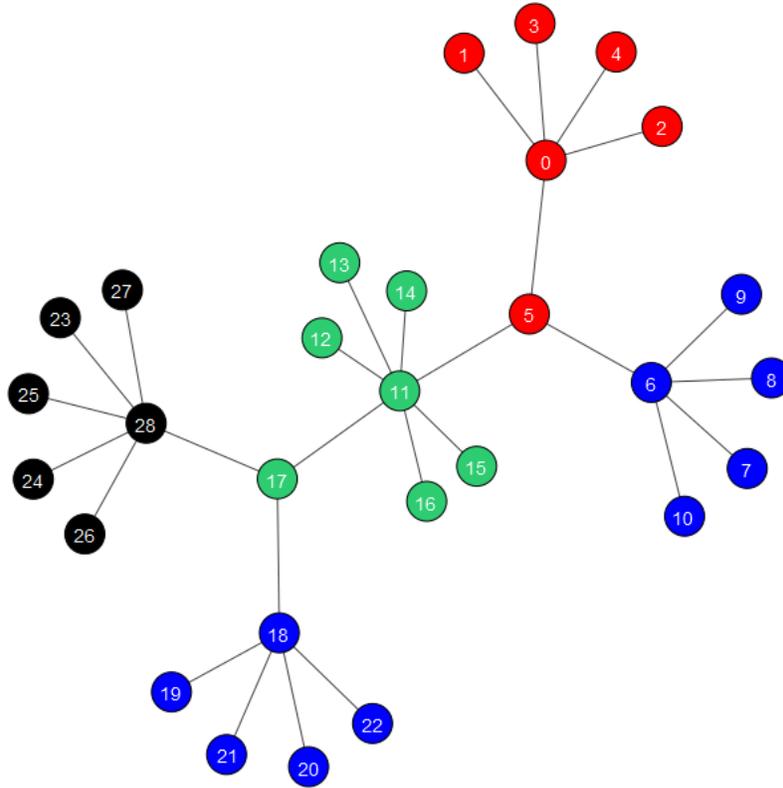


Figure 60: 5Comm attributed for kmeans with  $k = 4$

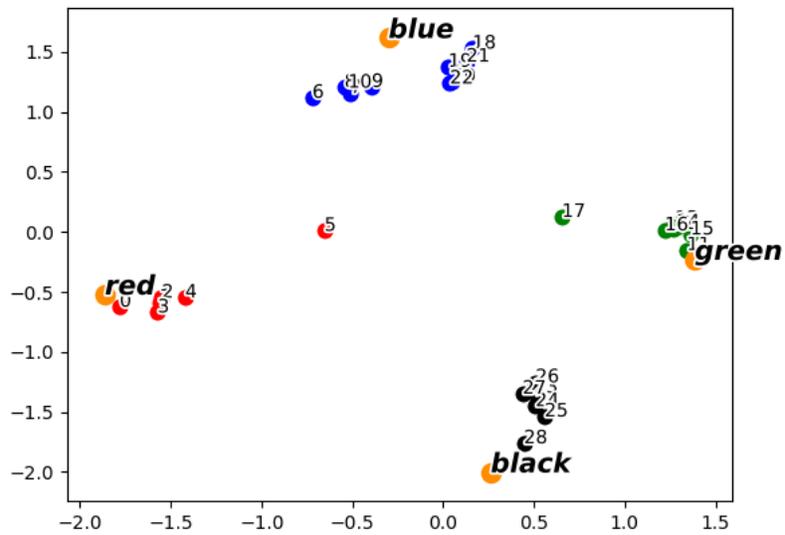


Figure 61: 5Comm attributed mds for k-means with  $k = 4$

Lowering the number of walks, i.e. the amount of data, can lead to more struc-

turally based results due to sub-sampling. This is because adding attributes makes it less likely for vertex ids to be subsampled, and as such node2vec-attr gets more vertex ids than node2vec with the same parameters. For example on one run of node2vec with a walk length of five and 100 walks vertex 11 has a frequency of 1459. The total amount of vertices in the random walks are 14500. then the probability of keeping vertex 11 is  $P(11) = 0.0011$ , where as if we add attributes to the walks we double the data an then  $P(11) = 0.1609$ . This causes it to find proximity. Lowering the number of walks for node2vec-attr will make it find structure.

This effect can be seen in Figure 62, which shows the attributed mds result when lowering the number of walks to 50. Here, the hubs of each community are more clearly separated from the rest of the vertices in that community, with the arguable exception of vertices 6 and 18 in the two blue communities. They are close to vertex 21, but they are still somewhat separated from the rest of the community. This indicates that node2vec-attr can distinguish between hubs and non-hubs, while still preserving the communities. Furthermore, the vertices of each community are now further away from their colour embedding, but for each color embedding, the community closest to it, is still the correct one. This suggests that node2vec-attr combines structural and semantic similarity.

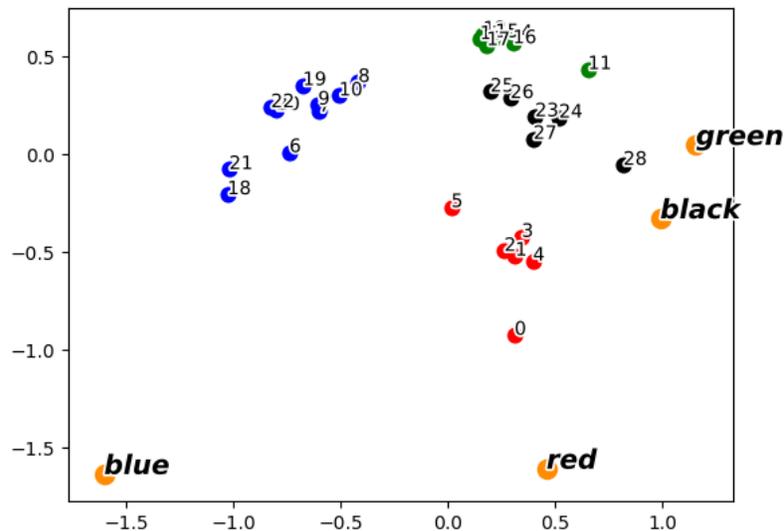


Figure 62: 5Comm attributed mds for k-means with  $k = 4$  for number of walks 50

### 9.2.1 Summary

When using parameters that produce structure based clustering for node2vec, node2vec-attr finds a mix between proximity and semantic based clustering. When decreasing the amount of walks node2vec-attr seems to combine structural and semantic similarity.

## 10 Case study: Airline network

To demonstrate the multirelational extension for node2vec, we performed a case study on the airline network using node2vec-multi. To recap, the network has 304 vertices, 3076 edges, and ten different relations representing the top ten airlines. In the walks of node2vec-multi, vertices are represented as follows:

$$\phi_{multi}(v) = id, relation$$

The two main focus points for this case study are:

- Do the airline embeddings provide useful information?
- Can you use the airport embeddings for a search task?

To answer the first question we visualize the embeddings of both the airport and airlines using mds. In addition, we investigate if any meaningful clusterings can be generated from the embeddings. To answer the second question we use the euclidean distance between embeddings as a similarity measure, and use this measure in a search task, to see if it provides any meaningful results.

Node2vec does not take multirelational graphs as input. We handle this by combining multiple edges into one edge and assigning it a weight based on how many edges were combined. So if there were three edges between two airports we replace this with a single edge with a weight of three.

### 10.1 Airline embeddings

In this section we examine whether the embeddings given by the multirelational extension of node2vec, referred to as node2vec-multi, provides any useful information. The parameters used for this analysis are:

$$dim = 128, wl = 80, numWalks = 10, windowSize = 3, p = 1, q = 4$$

Most of these are typical values for node2vec according to Grover et al. [6]. We decreased the window size from the default, to hopefully ensure that a vertex only appears in the context of close neighbours. For the same reason, we set  $p$  to one and  $q$  to four, which should keep the random walks local to their start vertices. We want to keep the walks relatively local to better capture local similarity.

We ran node2vec-multi with these parameters and applied mds to both the airline and airport embeddings, which is shown in Figure 63. The airport vectors are coloured blue, and the airline vectors are orange.

In general it does not seem like there is any clear clustering for the airports. It looks more like one large cluster, with the airlines located in the periphery of this cluster. Ryanair is an exception, as it is slightly closer to the centre, than the other airlines.

There are some airports which can be considered outliers. An example of this, is Trapani Airport(airport 69) in the left most middle (-1.8, 0.8), which only has

flights by Ryanair. There is also a small group of outliers on the right most middle (2.2,0.5). These airports are mostly smaller Scandinavian airports, such as Lycksele (airport 288) and Tromså (airport 252) airport. Both of these two airports are mostly serviced by SAS and Norwegian. This suggests that in general, airline embeddings will be close to embeddings of airports with many flights of said airline.

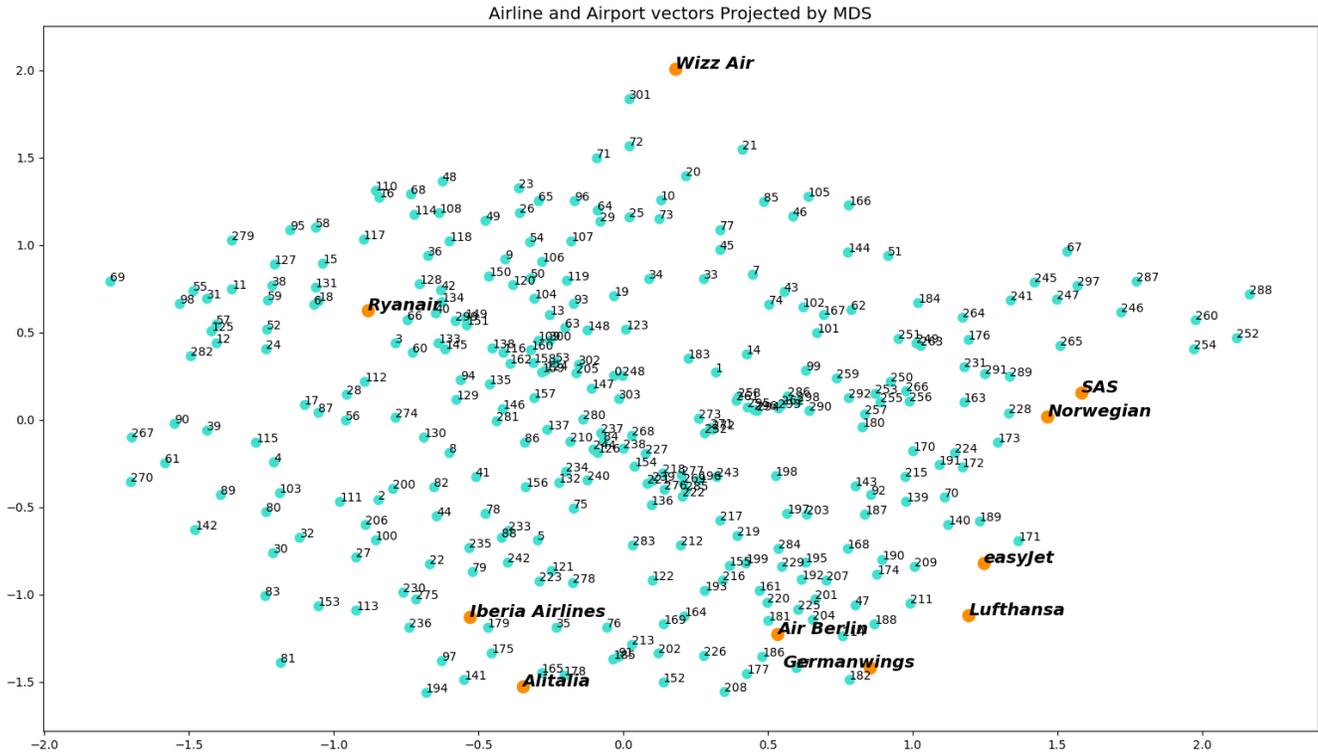


Figure 63: Multidimensional scaling of the embeddings given by node2vec-multi for airlines and airports

The airline embeddings seem to form four groups, see Table 8. It could be argued that easyJet and Lufthansa are in their own fifth group. Wizz Air appears to be an outlier because it lies far away from the other airlines, and isn't close to many airports either.

These groupings have inherent meaning. For example, it is sensible for SAS and Norwegian to be close together as these two airlines operate more heavily in Scandinavia than other airlines. Similarly, Lufthansa, Germanwings, and Air Berlin all tend to operate in central Europe, especially Germany. This suggests that our multidimensional extension, can create interpretable relation embeddings, which are potentially useful for data analysis.

	Airlines
Group 1	SAS, Norwegian
Group 2	Lufthansa, Germanwings, Air Berlin, easyjet
Group 3	Iberia, Alitalia
Group 4	Ryanair

Table 8: Airlines divided into groups based on their embeddings

To investigate whether there is a connection between the airlines and the airports, we used k-means with  $k = 4$  on the airport embeddings. The results are shown in Figure 64, where airports are coloured based on the cluster they are in.

As can be seen, it is more or less the case, that the airport clusters correspond to the airline groupings. For example, Ryanair appears to be almost in the centre of the blue cluster. It seems fair to assume, that airports in, for example, the Ryanair cluster should mostly have flights by Ryanair.

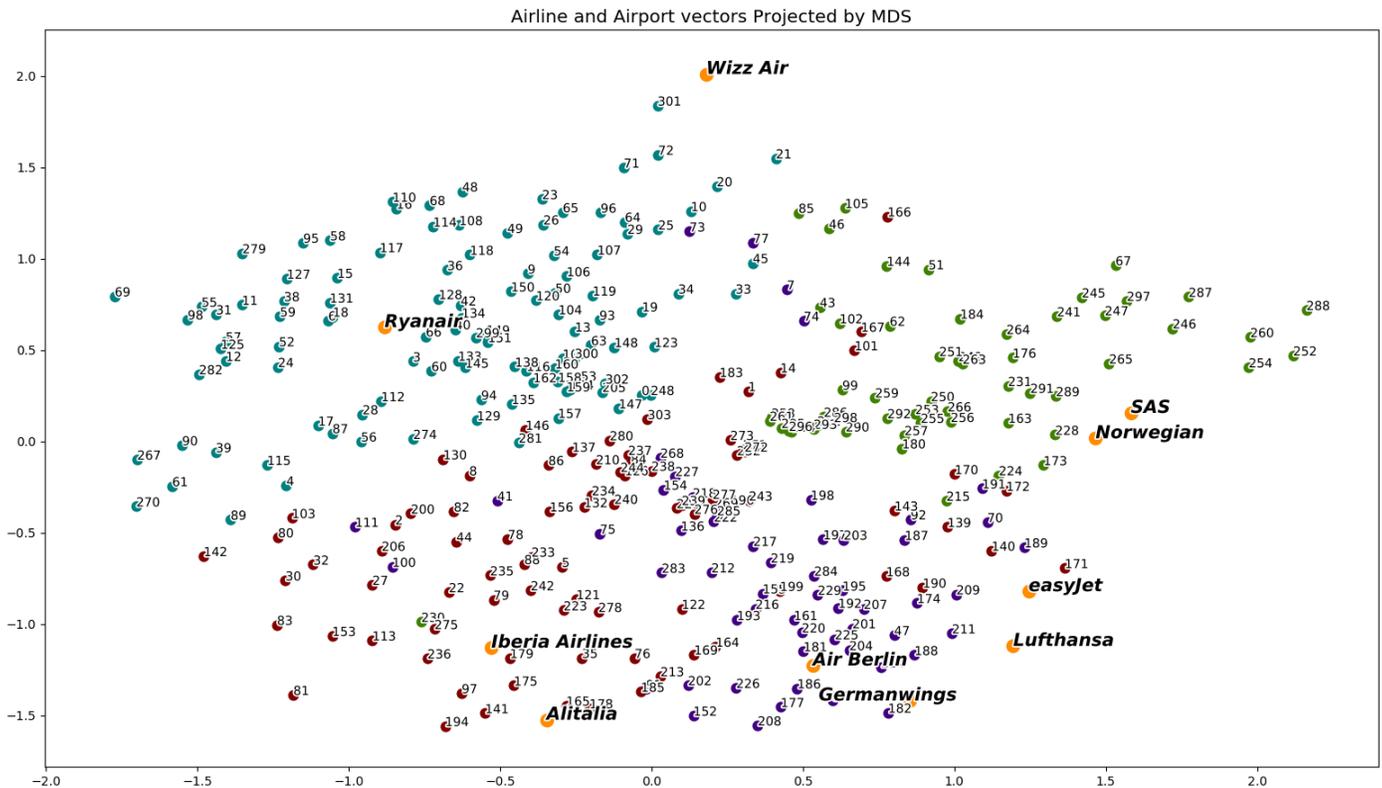


Figure 64: Multidimensional scaling of the embeddings given by node2vec-multi for airlines and airports. The colours correspond to the clusters created by k-means with  $k = 4$

To investigate whether this assumption holds we count, for each airline, the frequency of flights they have from the vertices in each cluster. For example, Norwegian has 300 flights from airports in the green cluster. This is seen in Figure 65.

In this figure, we can see the tendency that most airlines primarily operate in airports of one cluster. The airline embeddings are generally closest to the cluster in which they operate the most. For example, both Norwegian and SAS are closest to cluster 1(green), and the airports in this cluster have flights mainly by SAS and Norwegian.

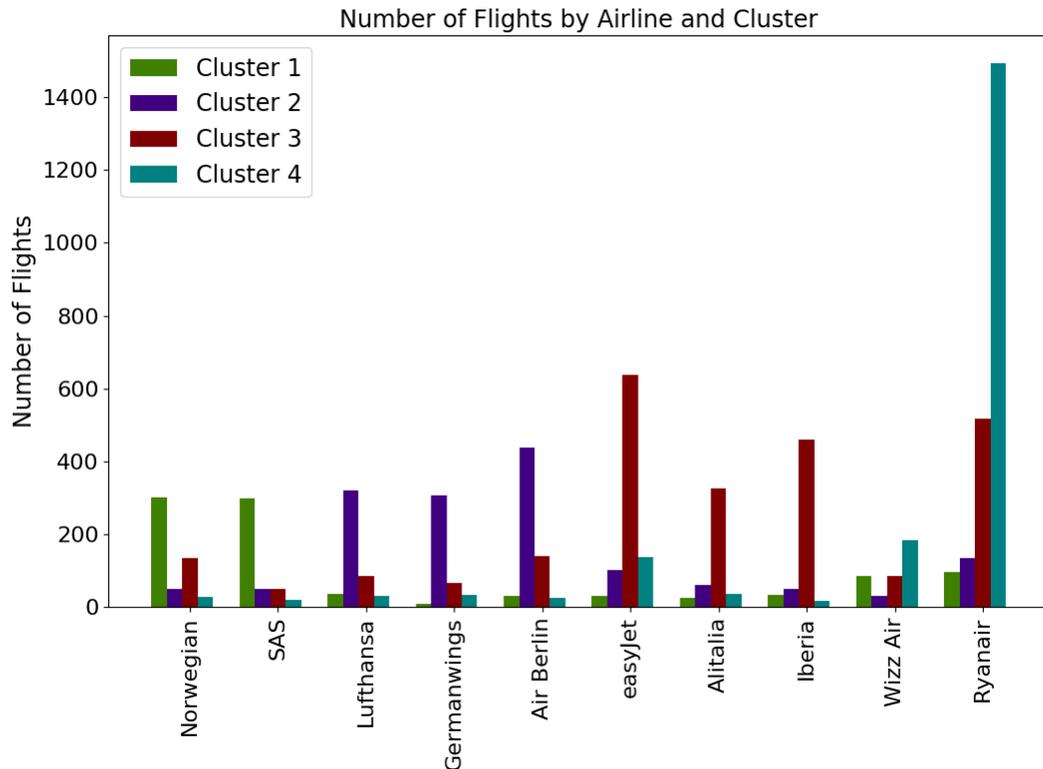


Figure 65: The number of flights by airline and cluster, when using node2vec-multi. Clusters correspond to the clusters shown in Figure 64

To compare node2vec with node2vec-multi we also ran node2vec with the same parameters, and clustered the embeddings using k-means with four clusters. The number of flights by airline and cluster can be seen in Figure 66. From the figure, we can to a certain extent observe the same pattern as in Figure 65. The largest difference is that easyjet and Alitalia now are mainly present in the same cluster as Lufthansa, Germanwings, and Air Berlin. This suggest that node2vec and node2vec-multi discover some of the same properties, but including relations can make some properties of the airports more clear.

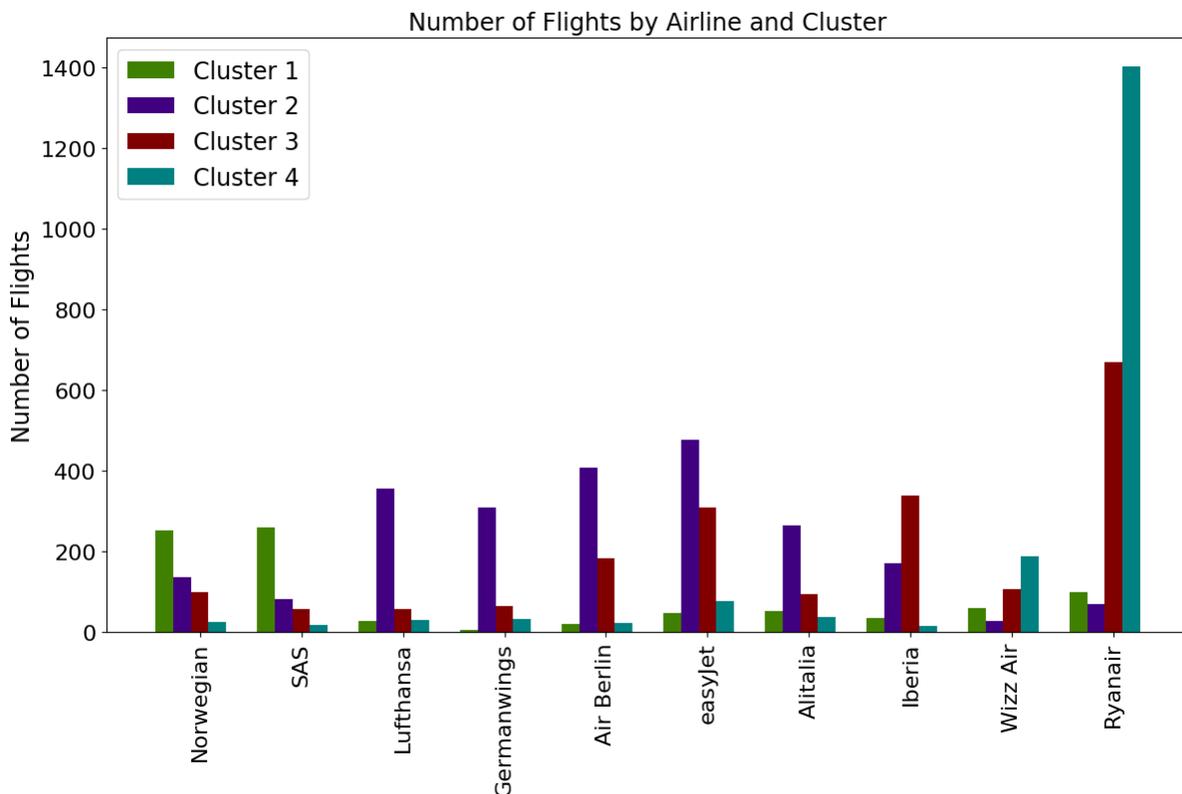


Figure 66: The number of flights by airline and cluster, when using standard node2vec

## 10.2 Airport search

The previous experiments were mainly focused on the airline embeddings. In this section, we will take a closer look at the airport embeddings. We do this by performing a search task, where we use the euclidean distance between airport embeddings as a similarity measure. For three query airports, we report the most similar airports given by the different node2vec versions. The three query airports are: Frankfurt am main Airport(Frankfurt), Václav Havel Airport Prague(VH Prague), and Norwich Airport. These three airports represent a hub, main-stream and periphery vertex respectively. Some basic statistics on the airports are given in Table 9.

	Degree	Nr of airlines	Airline with most flights
Frankfurt	116	6	Lufthansa: 89 of 116 (76,7%)
VH Prague	37	9	easyjet: 8 of 37 (21,6%)
Norwich	1	1	Alitalia 1 of 1 (100%)

Table 9: Query airport statistics

As was previously mentioned subsection 4.5.3, three kinds of context-target pairs are created when adding attributes to walks: vertex-vertex pairs, vertex-attribute pairs and type-type pairs. The situation is the same when adding relations. Too see

whether vertex-vertex and attribute-attribute pairs are needed to measure similarity we introduce `node2vec-multi2`, a version of `node2vec-multi` where only the vertex-relation pairs are used for training.

We do this by splitting the random walks up into walks of length two, each of which always consist of a vertex id and a relation. Note that from the formal formulation of Skip-gram, a sentence(or walk) would need to be of at least length three, but the specific implementation we used does not have this requirement.

For the search task we tested three different versions of `node2vec`: `node2vec`, `node2vec-multi` and `node2vec-multi2`. For all versions we used the same parameters:

$$dim = 128, wl = 80, numWalks = 10, windowSize = 3, p = 1, q = 4$$

The results are summarized in Table 10. For each query airport, we present the top three closest resulting airports. For each resulting airport, we show the Euclidean distance between the embeddings of the query and resulting airport, and the degree of the resulting airport. We are both interested in observing whether the extensions capture semantic similarity and whether any methods capture structural similarity

For Frankfurt, `node2vec-multi2` appears to give sensible results. The most similar airport it reports, is Munich, which makes sense, as Munich and Frankfurt are the two main hubs for Lufthansa, so they are both semantic and structurally similar. Düsseldorf is another major German airport, and while it is not a hub for Lufthansa, it still is the airport with third most Lufthansa flights. Zürich seems less sensible. One reason why Zürich is in the top three most similar, is likely because its neighbours are largely a subset of Frankfurt neighbours(25/36).

`Node2vec-multi` identifies Munich as the most similar airport. However, the other airports do not seem particularly sensible, as they both only have one flight. The flight is Lufthansa so in that sense they are weakly semantic similar to Frankfurt. We will go into further detail later on why `node2vec-multi` has a tendency to pick vertices with low degree.

`Node2vec` also appears to give less sensible results. It is the only method to not have Munich as the most similar. The reason `node2vec` reports these three airports as the most similar, could be because the neighbours of these airports are largely just a subset of the neighbours of Frankfurt. For example, out of Zagreb’s nine neighbours five are also neighbours of Frankfurt. In addition, all three results airports are directly connected to Frankfurt. This indicates that `node2vec` captures proximity.

Q Airport	node2vec-multi2		node2vec-multi		node2vec	
Frankfurt	Munich	0.3588 142	Munich	0.8990 142	Zagreb	0.2472 9
	Düsseldorf	0.7395 123	Rotterdam	1.1197 1	Larnaca	0.2611 14
	Zürich	0.7898 40	Sibiu	1.1284 1	Gothenburg	0.2877 23
VH Prague	Malpensa	0.4776 61	Pau Pyrénées	0.8556 1	Larnaca	0.3694 14
	Fiumicino	0.5093 118	Bolzano	0.8660 1	Athens	0.3888 32
	Amsterdam	0.5227 65	Norwich	0.8878 1	Frankfurt	0.4033 116
Norwich	Humberside	0.2169 1	Humberside	0.2289 1	Sundsvall-Timrå	0.1065 1
	Pau Pyrénées	0.2224 1	Durham Tees	0.2631 1	Kristiansund	0.1067 1
	Leon	0.2307 1	Pau Pyrénées	0.3594 1	Durham Tees	0.1094 1

Table 10: The top three results airports according to node2vec, node2vec-multi, and node2vec-multi2. For each result airport two properties are presented: The Euclidean distance between the airport embeddings and the degree of the resulting airport

For VH Prague node2vec-multi2, we will only go into further detail on Malpensa. There can be several factors why that Malpensa is picked as the most similar airport. One factor could be that they are directly connected, and share 17 neighbours. Another factor, is that they have a somewhat similar distribution of airlines among their flights: for example, they both have most flights by easyjet. The distributions are as follows:

	<i>AB</i>	<i>AL</i>	<i>easyjet</i>	<i>GW</i>	<i>LH</i>	<i>IB</i>	<i>NOR</i>	<i>SAS</i>	<i>Ryanair</i>	<i>Wizz</i>
<b>VH Prague</b>	( 0	5	8	3	2	3	5	3	2	6 )

	<i>AB</i>	<i>AL</i>	<i>easyjet</i>	<i>GW</i>	<i>LH</i>	<i>IB</i>	<i>NOR</i>	<i>SAS</i>	<i>Ryanair</i>	<i>Wizz</i>
<b>Malpensa</b>	( 3	10	32	5	3	3	1	2	0	2 )

This once again indicates that node2vec-multi captures semantic similarity. node2vec-multi once again has airports with a degree of one in its top three.

For node2vec we have a similar situation as with Frankfurt, in that the airports often have common neighbours.

All methods give reasonable results for Norwich. Recall that Norwich had one flight which was Alitalia. Both node2vec-multi2 and node2vec-multi pick airports with one Alitalia flight. Node2vec also finds airports with one flight, but as node2vec does not take into account airline information only one of the airports have an Alitalia flight. Interestingly, Humberside and Durham Tees are structural equivalent i.e.  $s_{jaccard}(Humberside, DurhamTees) = 1$ , because they are both only connected to Amsterdam airport.

From the table, it is clear that node2vec-multi will often pick airports with only one flight. This is appropriate for Norwich, but less so for VH Prague and Frankfurt. For all of these one flight airports, it is never the case that it is connected to the query airport. For example, both Sibiu and Rotterdam Hague are only connected to Munich. To see why node2vec-multi has this tendency of having one flight airports in the top three, we further examine the results for VH Prague. For the sake of simplicity, we primarily look into why Bolzano were in the top two results airports for VH Prague.

In Figure 67 we see part of the 2-neighbourhood of VH Prague, containing the following vertices: Charles De Gaulle, Fiumicino, Bolzano and Pau Pyrénées. The first observation to be made is that, while Bolzano and Pau Pyrénées are not directly connected to VH Prague, they are close with a distance of two.

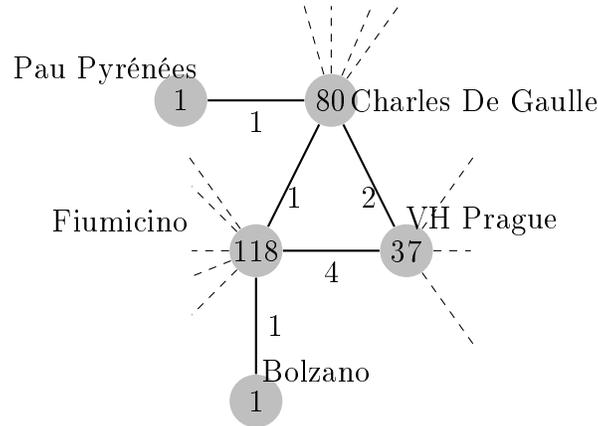


Figure 67: Parts of the neighbourhood of VH Prague. The weights of the edges are the number of flights. The degree of a vertex is written inside it

To understand the results of node2vec, it is important to further investigate the random walks. Recall that the random walks are used as input for the Skip-gram model. The Skip-gram model creates embeddings such that two words, or two vertices in our case, have similar embeddings when they often appear in the same context. Vertices with a degree of one will always have the same immediate context. For example, if we consider walks including Bolzano there are two cases. In the first case, it is the start vertex of the random walk, and in the second case, we go from Fiumicino to Bolzano. Example walks of the two cases are:

Case 1 : (*Bolzano, Alitalia, Fiumicino, ...*)

Case 2 : (*..., Fiumicino, Alitalia, Bolzano, Alitalia, Fiumicino, ...*)

This means that Bolzano will very often occur together with Fiumicino and Alitalia. VH Prague likely appear in more diverse contexts, but it should also occur somewhat often with Fiumicino and Alitalia. The reason that VH Prague often appears with Fiumicino, is that the edge going from VH Prague to Fiumicino have the highest weight among all incident edges on VH Prague. In addition, since almost half of the Fiumicino flights(57/118) are Alitalia, we could often have:

(*..., VHPrague, {easyjet, Alitalia, WizzAir, Iberia}, Fiumicino, Alitalia, ...*)

This would mean that Bolzano and VH Prague often appear in the same contexts(Fiumicino and Alitalia), and would therefore at least have somewhat similar embeddings, even though they might not appear together. The situation is much the same with Pau Pyrénées as half(40/80) of Charles de Gaulle’s flights are Alitalia.

To see whether the observations made for VH Prague and Bolzano regarding their context were correct, we further examined the frequency of words appearing in the context of the two airports, see Figure 68. The top row shows the frequencies of the ten most frequent context vertices given by node2vec-multi, while the bottom

row shows the frequencies given by node2vec. The left column shows the results for VH Prague, the right column for Bolzano. Alitalia is coloured orange and Fiumicino

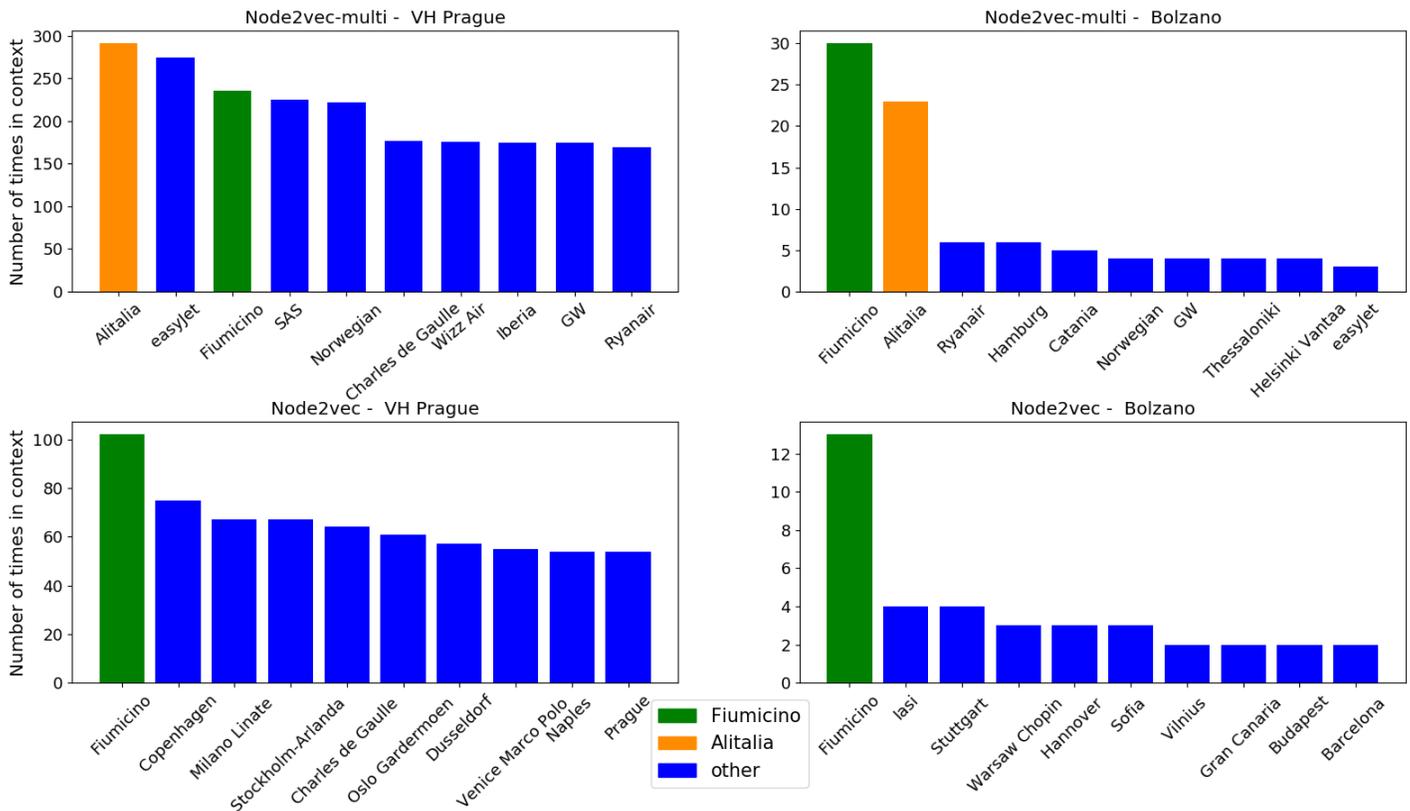


Figure 68: The frequency of context vertices for VH Prague and Bollzano. The left column is VH Prague and the right column is Bolzano. The top row is the contexts created by using node2vec-multi and the bottom row is the context created by node2vec.

The most apparent observation, is that Fiumicino and Alitalia appear much more frequently in the context of Bolzano, than any other airport. For VH Prague, it is the case that Alitalia and Fiumicino are the first and third most frequent context respectively, but they are not much more frequent than the other "words". As both airports often have Alitalia and Fiumicino as their context, they should at least be somewhat similar. However, VH Prague have many contexts that Bolzano does not have.

The reason why Bolzano is the most similar to VH Prague could then be that the specific contexts in which VH Prague appear are so unique that it essentially is only similar to itself.

In node2vec there are no airlines in the context, so it is only required that two airports have similar airport contexts, which is more lenient. For example, in the bottom row of Figure 68, we can see the context frequency for VH Prague and Bolzano. In this case, both airports appear mostly with Fiumicino, but as was seen in Table 10,

Larnaca airport was most similar to VH Prague. This indicates that they probably appear more often in the same contexts.

To summarize, there are several choices which need to be made, when using node2vec for a search task. The first, which we did not explore in detail, is choosing the parameters. The second is then to choose the version of node2vec to use. For this specific dataset it seems that node2vec-multi2 were the most sensible choice. We also showed that including relations in walks can help discovering semantic similarity. In addition to understanding the results of any node2vec version, one can gain additional insight by investigating the the context-target pairs created by Skip-gram.

## Part IV

# Experimental study

In this part we present an experimental study of node2vec and our extensions on real networks. The goal is to test how well our extensions hold up to node2vec in terms of performance and scalability.

We perform two classification tasks, namely, a multiclass and multilabel classification. The multiclass task is performed on the airline network, where the task is to predict the size of airports. To evaluate the effect of adding attributes to node2vec, we perform multilabel classification on three different networks, used by the authors of node2vec.

We evaluate the scalability for node2vec and two of our extension with attributes, by running them with default parameters on Erdos-Renyi graphs.

## 11 Experimental procedure

This section details the experimental procedure. It covers the vertex classification task, the evaluation measures used, and the methods used for the experiments.

### 11.1 Vertex classification

A vertex classification task can be summarized as follows. Given a training set  $\mathbb{D} = \{(v_1, c_1), \dots, (v_n, c_n)\}$  of labelled vertices and  $c_i \in C = \{c_1, c_2, \dots, c_m\}$  is a class label. We want to train a classification model  $\gamma$  that maps unlabelled vertices to classes.

In this section we describe the methods used for the classification tasks.

#### 11.1.1 K-nearest neighbours classification

k-nearest neighbours(kNN) classifies instances based on a similarity or distance measure. In our case we use euclidean distance, so the lower the distance between two instances, the nearer they are. When a new instance is classified, it is assigned to the most frequently represented class amongst its  $k$  nearest neighbours.

### 11.2 Evaluation measures

*This subsection includes some material from last semesters report: Vertex Similarity in Graphs*

For evaluating how well the similarity measures perform on the classification tasks, we use classification evaluation measures.

#### Classification evaluation measures

To evaluate the performance of our methods, we use F1-score. The F-score is a way of combining precision and recall to a single score, and can be seen in Equation 43. It

is primarily used as a way to compare the performance of different methods as it does not tell you whether the precision or recall are the cause of a high or low F-score.

$$F_{\beta} = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall} \quad (43)$$

The  $\beta$  constant can be used to weigh precision differently than recall. The traditional F-score(F1-score), which is the one we consider, uses a  $\beta$  value of one, weighing both precision and recall equally.

The F-score presented here is defined for binary classification tasks. For multiclass classification there exist different versions of the F-score, macro and micro F-score. The macro F-score is obtained by first calculating the precision, recall and f-score for each class, and then finding the mean. The micro F-score is obtained by globally counting all true positives, false positives and false negatives.

Precision measures the proportion of correctly classified positive instances, formally defined in Equation 44.

$$Precision = \frac{TP}{(TP + FP)} \quad (44)$$

where  $TP$  is the number of true positives and  $FP$  is the number of false positives

Recall measures the proportion of true positive instances retrieved, which can be seen in Equation 45.

$$Recall = \frac{TP}{(TP + FN)} \quad (45)$$

where  $FN$  is the number of false negatives.

### 11.3 Methods used

For the experiments we use the following methods:

- Majority: A simple method that always guesses the majority label. In the case of multi label instance with  $n$  labels, the top  $n$  most frequent labels are chosen. Ties are handled by randomly picking a label.
- Neighbour count(NC): A method that guesses the most frequent label among the neighbours of a vertex. In the event that all labels of the neighbours for a vertex are unknown, the majority method is used. Like the majority method the top  $k$  most frequent labels are chosen in the case of multi label instance with  $n$  labels. Ties are handled randomly.
- node2vec: node2vec is explained in greater detail in 4.5.2. We consider several of the extensions we proposed. When we run node2vec with attributes, we use the class label as the attribute. Furthermore, when training the model, the labels of the test data were set to "?".

- K-nearest neighbours+feature(KNN+gp): Classifies an instance based the k closets instances. What are the closets instances is based on a distance measure such as euclidean distance. In the +gp version, the vertices are represented by one or more simple vertex properties/statistics, such as degree or closeness centrality. For all experiments we set k= 5.

## 12 Dataset overview

A summary of the different datasets used for the experimental study is shown in Table 11.

	V	E	Avg. Degree	Diameter	Multirelational
BlogCatalog	10,312	333,983	64.78(sd = 177.70)	5	No
Wikipedia	4777	184,812	38.73(sd = 138.98)	3	No
PPI	3890	76,584	39.37(sd = 68.82)	8	No
Airline	304	3076	19 (sd = 24)	5	Yes

Table 11: Overview of the datasets used for the exploratory analysis

### 12.1 BlogCatalog

The BlogCatalog dataset is a social network containing information about bloggers and their blogs[21]. It consists of 10,312 vertices(bloggers), 333,983 edges(friendships), and 39 labels(groups). Here groups are categories a blogger can tag their blog with, such as art and fashion. One blog may be tagged with multiple groups. An edge exists between two bloggers  $b_1$  and  $b_2$  if  $b_1$  has included  $b_2$  to be part of his social network of other bloggers.

### 12.2 Wikipedia

The Wikipedia dataset[13] is a network of the cooccurrence of words appearing in the first million bytes of the Wikipedia dump. Labels in this network represent Part-of-Speech (PoS) tags inferred from the Stanford POS-Tagger. The network consists of 4777 vertices (words), 184,812 edges (cooccurrence), and 40 different labels (POS tags). An edge between two words exist if they co occur within a 2-length window in the Wikipedia corpus.

### 12.3 PPI

The Protein-Protein Interactions (PPI) dataset[5] is a network of interactions between proteins for Homo Sapiens. The network has 3890 vertices (proteins), 76,584 edges (biological interactions), and 50 different labels (biological states). Labels are obtained from hallmark gene sets.

## 13 Airline results

We do a multiclass classification task on the airline dataset. The purpose is to predict the size of airports, with the assumption that the size of an airport is highly dependent on the structural patterns of the vertices.

### Experimental procedure

We performed nine different train-test splits for each method, where the training data was incremented by 10% for each split, starting at 10%. For each train-test split, we run the experiments ten different times with different random seeds, and report the average results.

For the node2vec based methods, we learned the optimal  $p$  and  $q$  by training the methods on an 80-20 training-test split on 20% of the original labelled data for all possible combinations of  $p, q \in \{0.25, 0.5, 1, 2, 4\}$ . The optimal  $p$  and  $q$  values can be seen in Table 12.

	(p, q)
node2vec	(4, 1)
node2vec-attr	(1, 0.5)
node2vec-attr+	(0.25, 1)
node2vec-multi2	(0.5, 1)

Table 12: Chosen p and q values for node2vec representations

The other parameters were:

$$dim = 128, wl = 80, numWalks = 10, windowSize = 3$$

The embeddings produced by the different node2vec representations, were given as input to a knn with  $k = 5$ . The following methods will be used for the airline experiments: node2vec, node2vec-attr, node2vec-attr+, Neighbour count(NC) and Majority. In Table 13 is an overview of the different node2vec walk representations.

	Walk example
node2vec	(1, 29, 40, 1, 21)
node2vec-attr	(1 A, 29, B, 40, ...)
node2vec-attr+	(1, A, AAB, AB, 29, B ...)
node2vec-multi2	((1, R1), (1, R2), (2,R1))

Table 13: node2vec representations

In node2vec-attr and node2vec-attr+ we use class labels as attributes.

## Results

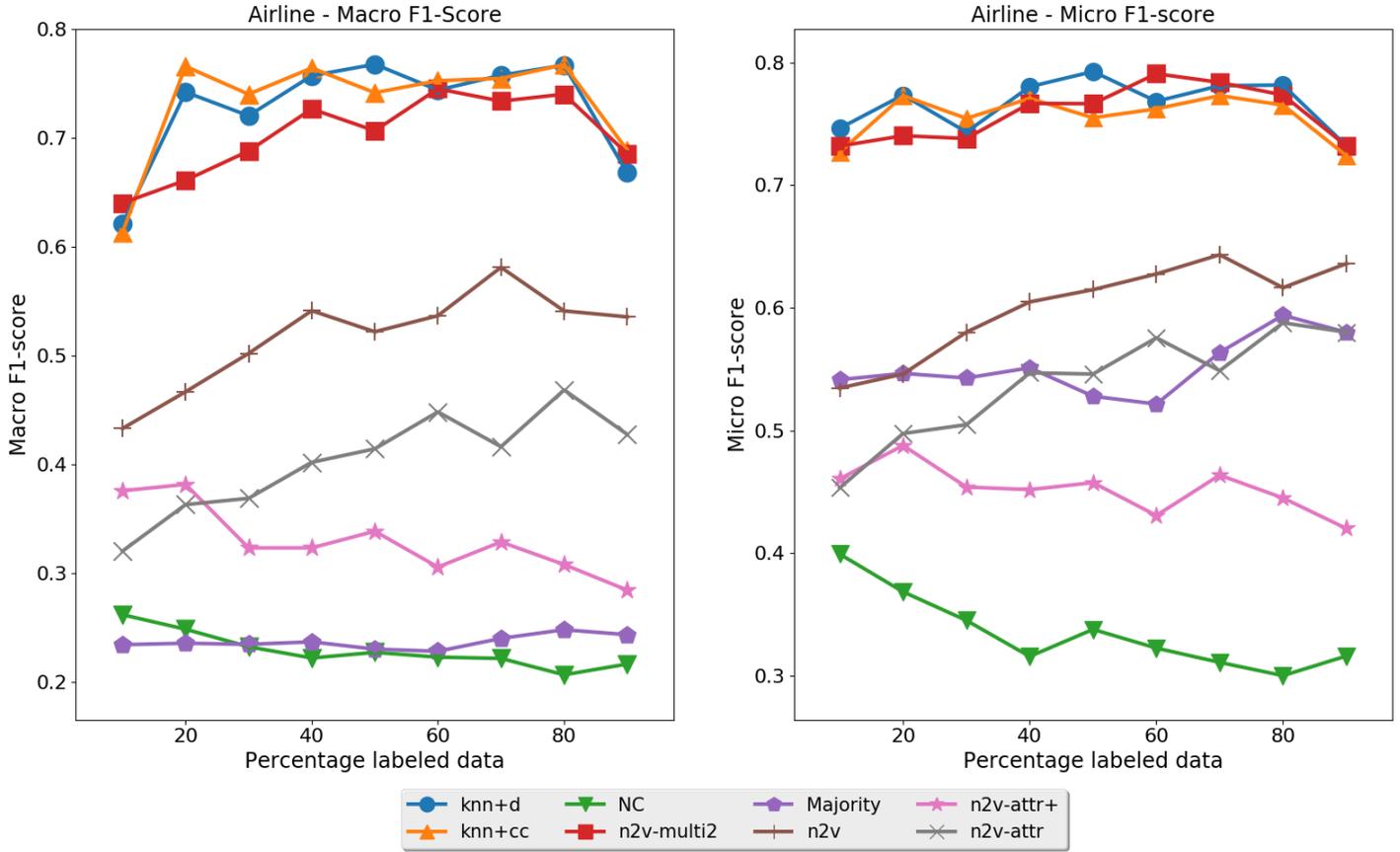


Figure 69: Results for the airline network

Figure 69 shows the results for the micro and macro F1-score for the airline dataset. Neighbour count does poorly both in regards to micro and macro F1-score. Majority achieves together with NC the worst macro F1-score, however, it achieves micro F1-scores comparable to the node2vec based methods. This is because of the class labels imbalance in the airline network. Almost all node2vec extensions were slightly worse than the original node2vec, with exception of node2vec-multi2 which is the best extension by a large margin. node2vec-multi2 together with knn+degree and knn+closeness-centrality are the best methods both in terms of macro and micro F1-score. knn+degree being among the best methods were expected, as in general small airports will have few flights(low degree), and large airports will have many flights(high degree). The reason that node2vec-multi2 was almost as good as knn+degree, could be because airports of similar degree are often mapped closer together, as we observed in the Case-Study.

## Discussion

From our exploratory analysis we showed that node2vec in general seems to just find proximity when the walk length or number of walks is increased. Therefore, we examined the effect of increasing the walk length and number of walks. In these experiments  $p$  and  $q$  are set to one. For the walk length experiment, number of walks were set to ten, and in the number of walks experiment, the walk length were set to 80. We use knn again for the classification. The result of the experiments are shown in Figure 70.

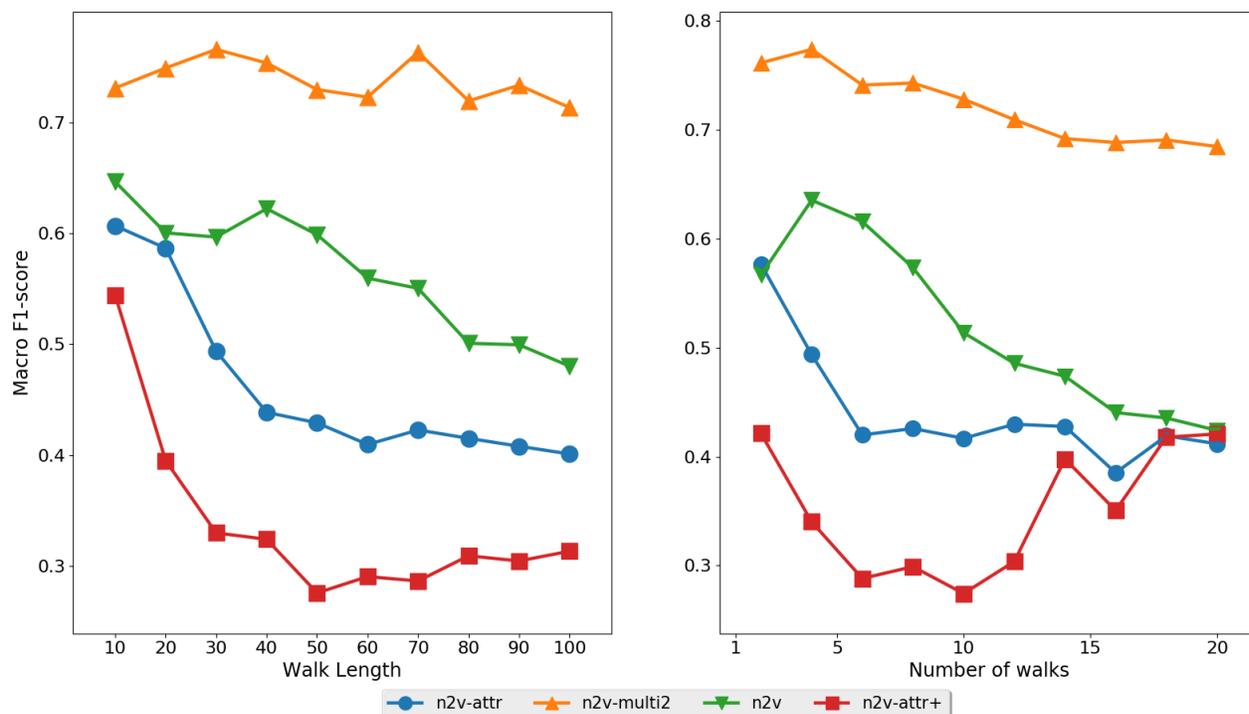


Figure 70: Performance of node2vec based methods when increasing walk length and number of walks. The  $y$ -axis denote the macro F1-score

We observe that the performance of node2vec and node2vec-attr tend to decrease, when we increase the walk length and number of walks. Both node2vec-attr+ and node2vec-attr have large drops in performance, going from ten to 40 walk length. This dive in performance is mirrored for the number of walks, where the dive happens from two to six. The performance of node2vec on the other hand decrease steadily. Interestingly, the performance of node2vec-attr+ improves slightly when increasing the number of walks. This slow increase might be because the vocabulary of node2vec-attr+ is larger than the other methods, and therefore require more data.

All node2vec methods, with the exception of node2vec multi2, obtain the best performance at low walk lengths(ten), and at low number of walks(one to two). This is most likely because, at larger walk lengths, we no longer find any structural similarity, just proximity, much like we discovered in the exploratory analysis.

node2vec-multi2 is fairly stable over increasing walk length, however, its performance decreases slightly when increasing the number of walks. The reason why node2vec-multi2 does well, could be because it only uses vertex-attribute(relation) context-target pairs. Because of this we also tested this approach for the other node2vec extensions. In addition, we also tested a version where we kept both vertex-vertex, and vertex-attribute context-target pairs. Finally, we also tested using only the individual feature for node2vec-attr+ i.e. all-neighbour and the unique-neighbours.

We used all these different methods on the airline dataset, with the default parameters from the airline experiments, however,  $p$  and  $q$  were both set to one. The training test-split were 50-50. The macro f1-score of these experiments are summarized in Table 14.

	n2v	attr	attr+	all-neighbours	unique-neighbours
All pairs	0.5136	0.4066	0.2877	0.2742	0.4806
vertex-vertex, vertex-attribute	-	0.5023	-	0.4563	0.5900
vertex - attribute	-	0.5828	-	0.3953	0.6957

Table 14: Performance of node2vec extensions when changing the the kind of context-target pair types used for learning

Using all pairs, the best method is node2vec, but unique neighbours is fairly competitive to node2vec. All-neighbours and node2vec-attr+ does far worse than all other methods. It seems that the generally poor performance of node2vec-attr+, were due to the all-neighbours feature. When only keeping the vertex-vertex and vertex-attribute pairs, the performance of all methods increase by a fair amount. Now node2vec-attr is more or less equal with node2vec, and unique neighbours is better. While the performance of all-neighbours is increased, it is still by far the worst method. When only using vertex-attribute pairs, the performance of node2vec-attr and unique-neighbours are increased. These two methods are now both better than node2vec.

So to summarize we found that our node2vec extensions were able to outperform node2vec, however, one needs to take several factors into account, such as what kind of pairs are allowed in the training and what features should be used in the walks.

## 14 Multilabel classification

To evaluate the effect of adding attributes to the node2vec walks we conduct a multilabel classification task on the datasets in the same manner as Grover et al. [6]. These datasets are BlogCatalog, PPI and Wikipedia.

### Experimental procedure

We performed three different train-test splits for each method: 20-80, 50-50, 80-20. The embeddings created by the node2vec methods were used as input into a one-vs-

	BlogCatalog	PPI	Wikipedia
node2vec	(0.5 ,1)	(4, 0.25)	(0.5, 0.25)
node2vec-attr	(4, 0.25)	(0.25, 2)	(0.25, 0.5)
node2vec-attr+	(1,1)	(4,4)	(4,2)

Table 15:  $p$  and  $q$  values for the different methods on the datasets

rest logistic regression classifier. Training instances with more than one label were duplicated one time for each label. For example if a vertex  $v$  had labels (1,2,3) there will be three training instances for  $v$ , each with the same embedding but with different class labels. For the test instances we assume that the number of labels is known beforehand, so for an instance with  $k$  labels the top  $k$  most probable labels are predicted. Regarding the node2vec extensions we view the attribute/type of a vertex as the concatenation of that vertex’s labels.

The optimal  $p$  and  $q$  were found by training the methods on an 80-20 training-test split on 10% of the labelled data for all possible combinations of  $p, q \in \{0.25, 0.5, 1, 2, 4\}$ . The optimal  $p$  and  $q$  values can be seen in Table 15.

For node2vec-attr and node2vec-attr+ we use the class labels as attributes. In case of multiple labels, they are treated as a single unique label.

## Results

In this section we present the results of performing multi label classification on the BlogCatalog, PPI and Wikipedia datasets.

The BlogCatalog results are summarized in Table 16. We can see that both Neighbour count and Majority are consistently the worst of the methods on both macro and micro F1-score. Majority being way worse than the rest. The different node2vec versions perform a lot better, with both of our extensions performing better than node2vec in terms of Macro F1-score for all percent of Labelled. However node2vec consistently achieves the highest micro F1-score.

	% Labelled vertices	20 %	50 %	80 %
Macro F1-score	node2vec	0.1913	0.2195	0.2310
	node2vec-attr	0.2070	0.2457	0.2677
	node2vec-attr+	0.2073	0.2479	0.2648
	Neighbour count	0.1276	0.1679	0.1887
	Majority	0.0254	0.02614	0.0271
Micro F1-score	node2vec	0.3627	0.3813	0.3924
	node2vec-attr	0.3406	0.3644	0.3842
	node2vec-attr+	0.3354	0.3731	0.3889
	Neighbour count	0.2296	0.2948	0.3239
	Majority	0.16468	0.1669	0.1719

Table 16: BlogCatalog macro and micro F1-score

The results for PPI are summarized in Table 17. We still notice the trend that Neighbour count and Majority perform the worst in terms of macro and micro F1-score. We can also see that all versions of node2vec perform about the same, with node2vec being slightly better in terms of micro F1-score in all configurations of labelled data.

	% Labeled vertices	20 %	50 %	80 %
Macro F1-score	node2vec	0.1511	0.1775	0.1904
	node2vec-attr	0.1444	0.175	0.19
	node2vec-attr+	0.1470	0.1736	0.1929
	Neighbour count	0.1079	0.1446	0.1668
	Majority	0.0171	0.016	0.0164
Micro F1-score	node2vec	0.1818	0.2088	0.2217
	node2vec-attr	0.1696	0.1957	0.2107
	node2vec-attr+	0.1699	0.1964	0.2169
	Neighbour count	0.1327	0.175	0.2038
	Majority	0.0667	0.0621	0.0631

Table 17: PPI macro and micro F1-score

The results for Wikipedia are summarized in Table 18. All the methods generally perform a lot worse on this dataset compared to the others, with Neighbour count and Majority still being the worst. Here we see that node2vec performs better than our extensions for both macro and micro F1-score.

	% Labeled vertices	20 %	50 %	80 %
Macro F1-score	node2vec	0.0915	0.1049	0.1156
	node2vec-attr	0.0707	0.0776	0.084
	node2vec-attr+	0.0768	0.0869	0.0925
	Neighbour count	0.0444	0.0457	0.0455
	Majority	0.0342	0.0337	0.034
Micro F1-score	node2vec	0.4873	0.5087	0.523
	node2vec-attr	0.416	0.4205	0.4375
	node2vec-attr+	0.4159	0.4357	0.4539
	Neighbour count	0.2334	0.2529	0.2967
	Majority	0.4119	0.4085	0.4159

Table 18: Wikipedia macro and micro F1-score

Generally it can be observed that the node2vec extensions perform better than NC and Majority by a large margin. On all datasets node2vec achieves the highest micro f1-score for all configurations. In terms of macro F1-score it differs from dataset to dataset which node2vec version is the best.

## Discussion

To obtain a better understanding of the results, we first examine the distribution of labels in the different networks, see Figure 71. The Wikipedia dataset is very imbalanced, with three labels(1, 8, 9) making up 59 % of all labels. In addition, 20 labels have a frequency of ten or lower. This imbalance explains the low macro F1-scores and the high micro F1-score for Wikipedia. The BlogCatalog dataset also appear to be somewhat imbalanced, while PPI is relatively balanced.

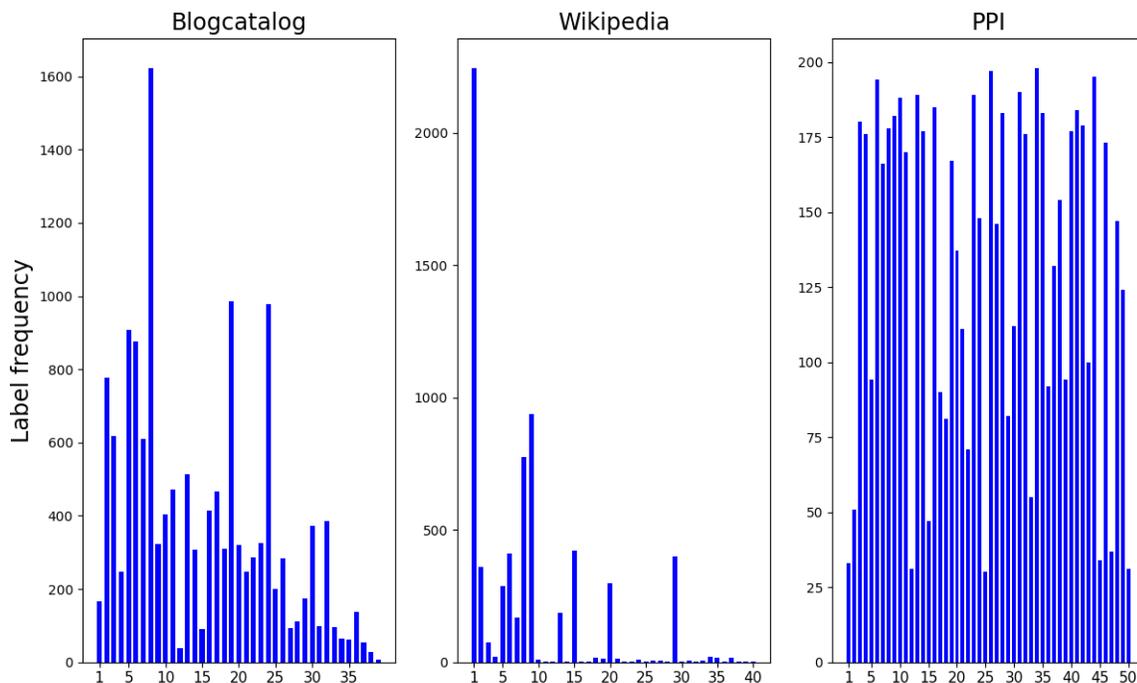


Figure 71: Distribution of class labels in the Blogcatalog, Wikipedia and PPI networks

To get a better understanding of why our extensions obtained a better Macro F1-score on the BlogCatalog data, we take a more detailed look at the F1-scores for the individual classes. One label that appears interesting, is label 27, where node2vec-attr obtained an F1-score of 0.1852, whereas node2vec obtained a score of zero. Label 27 appears a total of 94 times, 58 times as the only label and 36 times with at least one more label. To set these number in perspective, this means that they appear in 0.911 % of the vertices compared to the majority class, which appears in 15.73 % of the vertices. Even though there are so few instances, there are 17 vertices that have a label with 27 in it, and where the majority(or equal to the majority) of its neighbours have label 27. These vertices are shown in Appendix A. In some cases, it is not a very convincing majority, but nonetheless, these vertices should have label 27 in their context more often than other vertices in BlogCatalog. This is likely to be reflected

in the embeddings, resulting in the better F1-score.

While node2vec-attr and node2vec-attr+ did better on the BlogCatalog, it did worse on the Wikipedia dataset and comparable in the PPI dataset to node2vec. Recall that our strategy on how to represent a multilabel instance in node2vec-attr and node2vec-attr+, were to treat the multiple labels as one unique label. This may have been a too naive representation, especially for node2vec-attr+, as the features, all-neighbours and unique-neighbours, might become unique for every vertex. This would essentially mean that node2vec-attr+ becomes more or less the same as node2vec just with more data. In Table 19 we summarize how many unique labels are created for a single attribute, unique neighbours, and all neighbours.

	BlogCatalog	Wikipedia	PPI
Single attribute	1059	188	1063
Unique neighbours	9928	4775	3593
All neighbours	9993	4777	3613
Network size	10312	4777	3890

Table 19: Number of unique labels for single attribute, unique neighbours and all neighbours for the three datasets

It is evident from the table that our strategy to deal with multiple labels is not optimal. This is especially true for node2vec-attr+ as the additional features we add to the walks, are almost unique for every vertex for all datasets.

A better strategy to deal with the multiple label problem, could be to consider the multiple labels as a set of labels. So the unique neighbours feature for a vertex with two neighbours which have labels "1 2" and "1 3", will be "1 2 3" instead of "1 2 1 3". Using this approach the amount of unique labels are summarized in Table 20.

	BlogCatalog	Wikipedia	PPI
Unique neighbours	8639	4585	3562
All neighbours	9981	4776	3630
Network size	10312	4777	3890

Table 20: Number of unique labels for, unique neighbours and all neighbours for the three datasets if multiple labels are treated as a set

This approach only marginally reduce the number of unique labels. This low reduction is most likely due to the large number of classes in each network. Possible further solutions could be to consider the top k labels instead of every neighbour label.

## 15 Scalability

In this part we perform a scalability test of node2vec and our proposed extensions. We test three different configurations, id, attr and attr+. We focus on the scalability

of the learning part of the algorithm because they do not differ in the sampling part. We test with the default node2vec parameters on Erdos-Renyi graphs with increasing sizes from 100 to 100000 vertices, with a constant average degree of ten. Each graph size is run ten times, and the average time is taken. The hardware used to run the experiment has the following specifications:

CPU: Intel(R) Core(TM) i7-2600 CPU @ 3.40 GHz

Memory: 16,0 GB RAM

OS: Win 7 64 bit

Python 2.7.13

In Figure 72 we see that the three versions all scale linearly with the number of vertices. We also see a small constant overhead for the attr+ version.

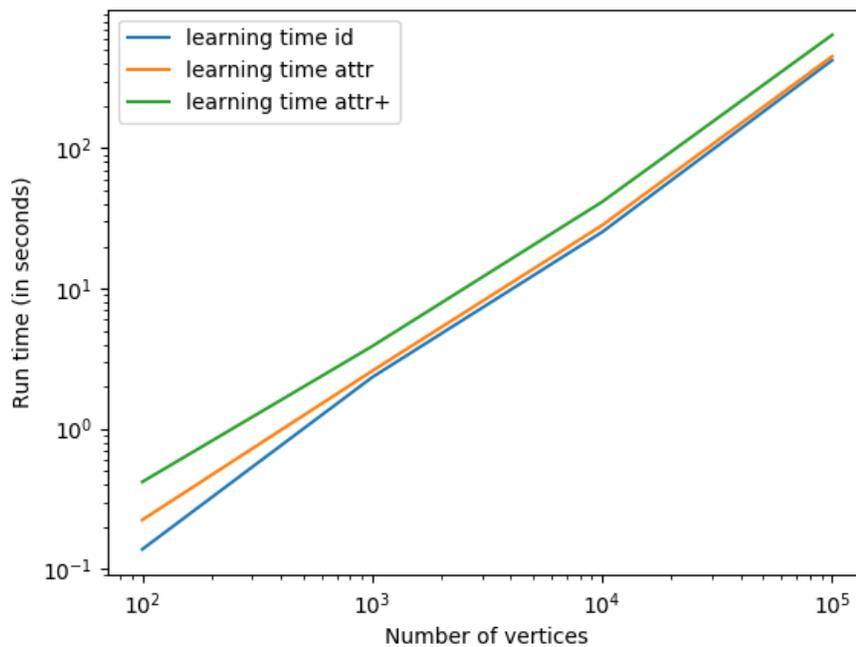


Figure 72: Scalability of node2vec on Erdos-Renyi graphs with an average degree of 10

## Part V

# Conclusion

We investigated the problem of measuring similarity between vertices in graphs. There are various different notions of similarity that can be measured on vertices, among these are: structurally, proximity and semantically based similarity.

State of the art techniques utilize feature learning, which objective is to learn embeddings of vertices. These embeddings could be used to determine the similarity between vertices. One novel feature learning technique is node2vec, which the authors claim is able to find either structure or proximity depending on parameters  $p$  and  $q$ .

We examined these claims and found that node2vec could find both structure and proximity, however, the parameters  $p$  and  $q$  had little to no effect on this. Instead, this was based on the walk length and the number of random walks.

Additionally, we proposed multiple extensions for node2vec, which allows it to use the attributes of vertices and edges. We perform experiments with these extensions on classification and search tasks and evaluate how they compare to node2vec. We found that for multiclass classification on an airline network, at least one of our multirelational extensions performed better in terms of F1-score than node2vec.

For future work, many directions can be considered. It would be interesting to further investigate why node2vec can only find structure at lower number walk length and number of walks.

Additionally, our extensions for attributes only work for discrete values, ways of handling continuous attributes could be interesting as many real networks contain attributes with continuous values.

## Bibliography

- [1] eurostat. <http://data.europa.eu/euodp/data/dataset/43C6uGqWp92dX7vlgNzJA>.
- [2] Openflight. <http://openflights.org/data.html>. Accessed: 2010-09-30.
- [3] Stefano Boccaletti, Ginestra Bianconi, Regino Criado, Charo I Del Genio, Jesús Gómez-Gardenes, Miguel Romance, Irene Sendina-Nadal, Zhen Wang, and Massimiliano Zanin. The structure and dynamics of multilayer networks. *Physics Reports*, 544(1):1–122, 2014.
- [4] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. IEEE, 2005.
- [5] Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, Michael Livstone, Rose Oughtred, Daniel H Lackner, Jürg Bähler, Valerie Wood, et al. The biogrid interaction database: 2008 update. *Nucleic acids research*, 36(suppl 1):D637–D640, 2008.
- [6] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*, pages 855–864, New York, NY, USA, 2016. ACM.
- [7] Xionglei He and Jianzhi Zhang. Why do hubs tend to be essential in protein networks? *PLoS Genet*, 2(6):e88, 2006.
- [8] Mikko Kivela, Alex Arenas, Marc Barthelemy, James P Gleeson, Yamir Moreno, and Mason A Porter. Multilayer networks. *Journal of complex networks*, 2(3):203–271, 2014.
- [9] Donald Ervin Knuth. *The Stanford GraphBase: a platform for combinatorial computing*, volume 37. Addison-Wesley Reading, 1993.
- [10] Elizabeth A Leicht, Petter Holme, and Mark EJ Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006.
- [11] X. Li, H. Chen, J. Li, and Z. Zhang. Gene function prediction with gene interaction networks: A context graph kernel approach. *IEEE Transactions on Information Technology in Biomedicine*, 14(1):119–128, Jan 2010.
- [12] Zemin Liu, Vincent W Zheng, Zhou Zhao, Fanwei Zhu, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. Semantic proximity search on heterogeneous graph by proximity embedding. 2017.
- [13] M. Mahoney. Large text compression benchmark, 2011.
- [14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

- [15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [16] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, November 2011.
- [17] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, volume 5, pages 488–495, 2009.
- [18] Koji Tsuda and Hiroto Saigo. *Graph Classification*, pages 337–363. Springer US, Boston, MA, 2010.
- [19] Xifeng Yan, Philip S. Yu, and Jiawei Han. Graph indexing: A frequent structure-based approach. pages 335–346, 2004.
- [20] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.
- [21] Reza Zafarani and Huan Liu. Social computing data repository at asu, 2009.

## Appendix A

	Degree	Nr of neighbours with label 27	Proportion of neighbours with label 27
75	6	1	0.1667
5201	2	1	0.5
612	3	1	0.3333
132	1	1	1
8862	61	5	0.0820
6840	61	7	0.1147
9939	13	4	0.3077
2263	4	1	0.25
6283	2	1	0.5
4382	15	2	0.0667
3208	5	2	0.4
10071	12	6	0.5
3485	17	7	0.1176
3003	67	2	0.1044
9663	9	2	0.2222
4576	9	4	0.4444
2038	5	1	0.2

Table 21: The 17 vertices where label 27 is among the majority of neighbour labels