Finding Representative Entities From Entity Graph By Using Neighborhood Based Entity Similarity

by

Ankit Anil Shingavi

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington

in Partial Fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

Supervisor: Dr. Chengkai Li

August 2018

Abstract

Several applications deploy the use of large entity graphs. Given the entirety of its application scope, it is challenging to select a single entity graph for a particular need from numerous data sources. For a comprehensible overview of the entity graph, we may project a preview table for compact representation of an entity graph. Each preview table represents a single entity type in the dataset. We need to find the representative entities for a given entity type from the entity graph to show the coverage of a dataset.

In this paper, we propose a method to find representative entities for a given entity type from the entity graph. Each entity of the same type is represented by a multi-dimensional label vector using neighborhood nodes. We apply the k-means clustering algorithm on the generated label vectors of the same entity type. The clustering algorithm divides a set of entities into k disjoint clusters. The nearest entity to the centroid of each cluster is used as the representative entity for the given entity type.

We have performed experiments on the Freebase dataset, based off of which, we got diverse and important representative entities for the tv, film and location domain. We can use these representative entities in the generation of preview tables.

Finding Representative Entities From Entity Graph By Using Neighborhood Based Entity Similarity

ANKIT SHINGAVI, The University of Texas at Arlington
SUPERVISED BY: DR. CHENGKAI LI, The University of Texas at Arlington
COMMITTEE MEMBER: DR. RAMEZ ELMASRI, The University of Texas at Arlington
COMMITTEE MEMBER: DR. BOB WEEMS, The University of Texas at Arlington

Several applications deploy the use of large entity graphs. Given the entirety of its application scope, it is challenging to select a single entity graph for a particular need from numerous data sources. For a comprehensible overview of the entity graph, we may project a preview table for compact representation of an entity graph. Each preview table represents a single entity type in the dataset. We need to find the representative entities for a given entity type from the entity graph to show the coverage of a dataset. In this paper, we propose a method to find representative entities for a given entity type from the entity graph. Each entity of the same type is represented by a multi-dimensional label vector using neighborhood nodes. We apply the k-means clustering algorithm on the generated label vectors of the same entity type. The clustering algorithm divides a set of entities into k disjoint clusters. The nearest entity to the centroid of each cluster is used as the representative entity for the given entity type. We have performed experiments on the Freebase dataset, based off of which, we got diverse and important representative entities for the tv, film and location domain. We can use these representative entities in the generation of preview tables. This helps the data worker understand the coverage of a particular entity type in the dataset.

1 Introduction

An entity graph is a collection of nodes and edges. A node in an entity graph represents the entity and an edge represents the relationship between two entities. Usually, an entity graph is stored in the form of a Resource Description Framework (RDF) triplet. An RDF triplet consists of three elements: a subject, a predicate, and an object. The subject is a real-world entity and so is an object. The predicate represents a relationship between the subject and the object. There are several real-world entity graph datasets available such as Freebase[3], DBpedia[4], YAGO[5].

A more modern use of entity graphs is to often store real-world information. Fig. 1. represents an excerpt of an entity graph related to the film database. FILM, AWARD, FILM DIRECTOR are the entity types in the film domain. ‘Men In Black’, ‘Will Smith’, ‘Robot’ are the real world film entities represented as nodes in an entity graph of the film domain. An edge in the entity graph is the predicate from an RDF triplet. An edge between two entities indicates a relation between them. For example, an edge ‘Actor’ between ‘Will Smith’ and ‘Men In Black’ indicates that ‘Will Smith’ has acted in the film ‘Men in Black’ [2]. In an entity graph, two entities can be connected by multiple relations. For example, the entities ‘Will Smith’ and ‘Robot’ are connected by relations ‘Actor’ and ‘Executive Producer’.

It is challenging to select the entity graph for a particular need, given the abundance of datasets from myriad sources. More often than not, enough data is unavailable. [2]. One cannot get a direct look at an entity graph before fetching it. Downloading a dataset and loading it into a database can be daunting [2]. So, the preview tables described in [2] help the data worker have a look at actual data before using it. A domain data is described by many preview tables. Each preview table depicts information about single entity type in that domain. An entity type represents a class for the collection of entities. A single entity can have more than one entity type. In this way, we can understand the dataset by looking at the preview tables of that domain.

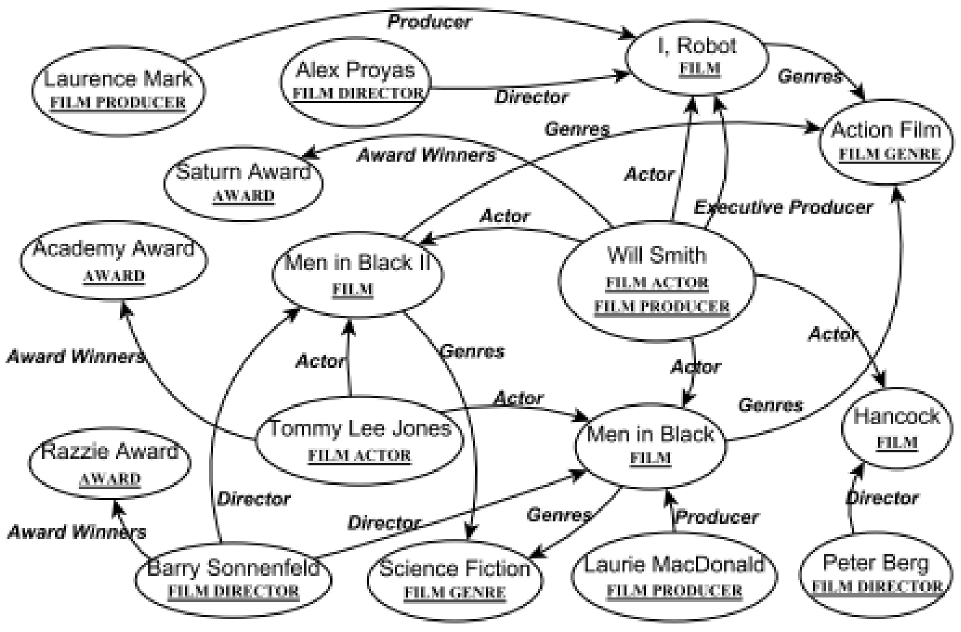


Fig. 1. Entity Graph
[2]

Finding representative entities from the graph is vital for representing a short preview table described in [2] of the larger entity graphs. Each preview table represents information about the single entity type. We need to find sample representative entities of the same type to make the preview table more informative. One way to select sample examples for a preview table is to choose entities randomly. Oftentimes, randomly selected entities do not represent the diversity of the entity type. Also, the random method can generate entities which are not important or representative among the entities of the same type. So, it is important to find the representative entities of the same entity type.

In this paper, we propose a method to find representative entities by employing the k-means clustering algorithm on entities of the same type. We represent each entity as the multi-dimensional vector called a label vector. A label represents a single dimension of the label vector. A label can be another entity or a relation from the entity graph of the same domain. A label vector of an entity is generated using the Information Propagation Model described in [1]. A label vector outlines the neighboring information of an entity in the entity graph. The cosine similarity function is used to calculate the similarity between two label vectors. Once we generate the label vector for each entity of the same type, we can apply k-means clustering to group them into the different clusters. The centroid of each resulting cluster is used as a representative entity sample for that entity type. Thus, we get diverse and important entities to represent an entity type in the preview table. The data worker can get an overview of the coverage of an entity type by looking at the representative entities in the preview table. It will help choose the right dataset for their requirement.

2 Problem Statement

Large entity graphs are used in most wide scaled applications. Considering the scope of the application and the gravity of the training data, it was challenging to identify the single appropriate entity graph from an extensive number of sources, which would best suit the application needs.

This challenge was met with a solution of preview tables [2]. A preview table helps the data worker better gauge the dataset with respect to its understanding and compatibility of the application under consideration. Each preview table represents a single entity type in the dataset, which allows a peek on to the dataset about to be used.

However, the solution could be better optimized by introducing a larger range of variation in the preview table data scope. Similar entities in the preview table would not be the ideal dataset and were not accurate enough to support decisions.

Thus, there was a further need of improvisation to identify representative entities for a given entity type from the entity graph to consider a holistic coverage of a dataset. The following method aims to propose a solution to find and portray representative entities for a given entity type from the entity graph.

Given an entity graph, we propose a method to find k representative entities from the entity graph for an entity type T.

$$F(G, k, T) - - > [e_1, e_2, e_3, ..., e_\mathrm{k}] \tag{1}$$

Where,

G is an entity graph represented in RDF format. An entity graph is a collection of nodes and edges. An entity graph represents real-world information about a particular domain. k is an integer number. The number k is the required number of representative entities of the given type. k is selected by the user based on requirements. Based on this number, we generate k number of representative entities.

T is the given entity type. The preview table represents the single entity type from the entity graph. We apply the algorithm to all entities of the given type T to generate k representative entities.

e_1, e_2, e_3, ..., e_k are the resulting representative entities of type T. Each representative entity of the given type is the nearest entity to the centroid of the resulting cluster.

The task is to generate k representative entities of type T from given entity graph G by using a neighborhood-based similarity approach.

3 Overview Of Method

With the advent of data science, the potentials of leveraging data onto making decisions have improved manifold. The topic of research under consideration, 'Neighborhood-based entity similarity' was coined in retrospect of traditional entity graph optimization.

We aimed to implement an algorithm that would find representative entities for a particular type from the given entity graph. This was an improvement from a random selection of entities, to a more educated and holistic view of the similar preview of entities that could be employed in supporting decisions. The applications could be used in many data science projects which require a decision support base.

The approach to this was adopted by applying clustering to the entities of similar type to obtain different groups. The centroids used to form a cluster act as representative entities that help build views that could be further used for real-time applications. The process of finding representative entities from a given entity graph by using neighborhood-based entity similarity includes 5 major steps. Unlike the previous method that would sample entities for a preview table based on a random selection, this approach aims to consider the complete scope and select the most accurate set of entities that would prove to be a more efficient result.

Finding representative entities from the entity graph for a given type is a five-step process. First, the program fetches required entity graph data from the database. The entity graph data is stored in an RDF format with the entity type as extra information along with the RDF triplet. Each row in the relational database table for an entity graph consists of five values: subject, subject type, predicate, object, object type. The subject type is an entity type for the subject. The object type is an entity type for the object. We store the entity graph data fetched from the relational database into a python nested dictionary of sets. The entity types for each entity is stored in the python dictionary format. Second, we extract all entities of a given type T using the stored dictionary of the entity types.

The next step is to generate label vectors for each extracted entity of a given type. Label vector is the multi-dimensional vector space portrayal of an entity. Label vectors are generated using the Information Propagation Model described in [1]. The information propagation model suggests that the information is propagated from the neighboring entities and relations with exponential decay [1]. So, we use neighboring entities and relations of an entity to represent it in multi-dimensional vector space. The label vector of an entity depicts the neighboring information around that node. We generate label vector for an entity using breadth-first traversal. Label vector generation process is described in details in section 4.

After generating label vectors for entities of a given type from the entity graph, we apply an unsupervised K-Means clustering algorithm to divide them into k sets. In the assignment step of the k-means clustering algorithm, we use the cosine similarity function to calculate the similarity between the label vector of an entity and the label vector of the centroid. After a few iterations, all entities of the same type are divided into k distinct clusters. Each cluster has a centroid which is calculated by taking an average of all label vectors of the entities within a single cluster. The centroid label vector of the resulting cluster may not be the actual real-world entity. So, we find the nearest entities to the resulting centroids and use them as the representative entities of the given type. These representative entities can be used for showing preview table for that type. We need to apply this algorithm for each entity type in the entity graph.

4 Label Vector Generation

A label can be either an entity or a relation (edge). Label vector of an entity in a graph is a multi-dimensional vector space representation of an entity. The values in a vector representation of an entity are weights given to n-hop neighborhood labels of an entity.

4.1 Information Propagation Model

The information propagation model described in [1] is used to generate label vectors. Based on the Information Propagation Model, information propagates along the shortest path between a label and an entity with an exponential decay to the length of a label from that entity [1]. This model suggests that we must emphasize the information propagated from the labels which are near to the given entity.

Fig. 2. represents the propagation of neighborhood information around entity node a. In Fig. 2., a, b, c, d, e are entities in a graph and r1, r2, r3, r4 are relations between entities. Arrow lines in figure show flow of the information from the neighboring nodes and edges to an entity 'a'. The weight for each label in the label vector is assigned using information propagation weight function based on a distance between the label and given entity.

4.2 N-hop Neighborhood Labels

All nodes and the edges which are at a maximum n-hop distance from the given entity are considered as n-hop neighborhood labels for that entity [1]. The n-hop neighborhood labels represent surrounding information about the entity in an entity graph. There are two types of the neighborhood labels: node labels and edge labels. The nodes in the n-hop neighborhood labels are the node labels. The edges or the relations in the n-hop neighborhood

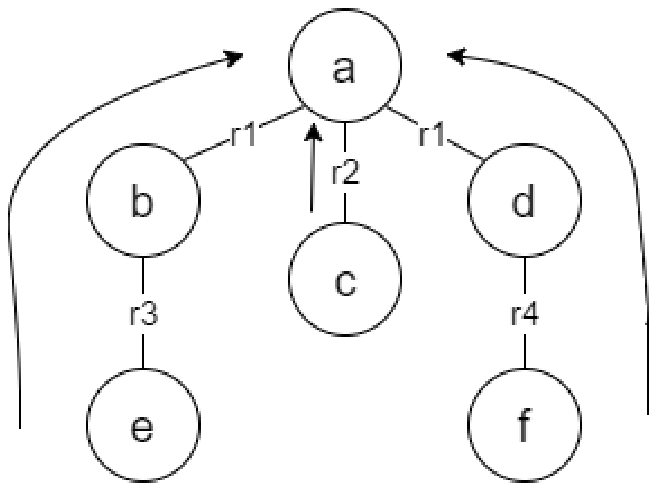


Fig. 2. Information Propagation Model

labels are called edge labels. For example, in Fig. 2., (b, c, d, r1, r2) set represents 1-hop neighborhood labels for an entity ‘a’. The nodes b, c, d are the node labels for entity ‘a’. The edges ‘r1’ and ‘r2’ are the 1-hop edge labels for the entity ‘a’.

4.2.1 Information Propagation Weight Function

The information propagation weight function is used to calculate the strength of each label in the vector space representation of an entity. This function uses the distance of a label from the given entity and the number of occurrences of a label in the n-hop neighborhood of given entity to calculate the weight of label in the label vector of that entity [1].

Equation 1 in [1] defines the formula for calculating the strength or weight of each label for the label vector of a given entity. This equation can be reformulated as follows:

$$Weight(e, l) = \sum_{i=1}^{n} \alpha^{i} * N_{\mathrm{i}}(e, l) \tag{2}$$

Where, α is the information propagation constant described in [1]. $N_{\mathrm{i}}(e, l)$ represents number of occurrences of label l at exactly i hop distance from an entity e.

The research work in [1] only considers the neighboring nodes as labels for their experiments. But in our experiments, we use neighboring nodes and edges both are considered as a label for formulating label vector for each entity. So, we have defined the formula for finding the weight for node labels and edge labels.

Weight Function For Node Labels:

We have modified the weight function defined in [1]. In our dataset, we have a unique id for each node in the entity graph. So, there will be only one occurrence of a node label in the neighborhood of an entity. So, we don’t need to find the number of occurrences of the node label in each level from the given node. The following equation represents generic information propagation weight function to calculate the weight of a node label(l) for the label vector of entity(e).

$$Weight(e, l) = \alpha^i \tag{3}$$

Where, α is a information propagation constant described in [1], i is the minimum hop distance from node label l from entity e. Information propagation constant (α) is a user defined factor which remains same for every node label in a graph.

Weight Function For Edge Labels:

The weight function for an edge label is modified from the original weight function defined in [1]. We can not use the same information propagation constant for edge labels because there can be multiple edges of same type present in the neighborhood of an entity. Using the same information propagation factor for all edge labels can result in assigning a higher weight to edge labels. We, thus, calculate the information propagation factor separately for each edge label. Information propagation factor for each edge label is calculated using the following equation.

$$\alpha_{\mathrm{l}} = \min(\alpha, 1/(\textit{max number of label l present in 1-hop neighborhood of any entity in given graph})) \tag{4}$$

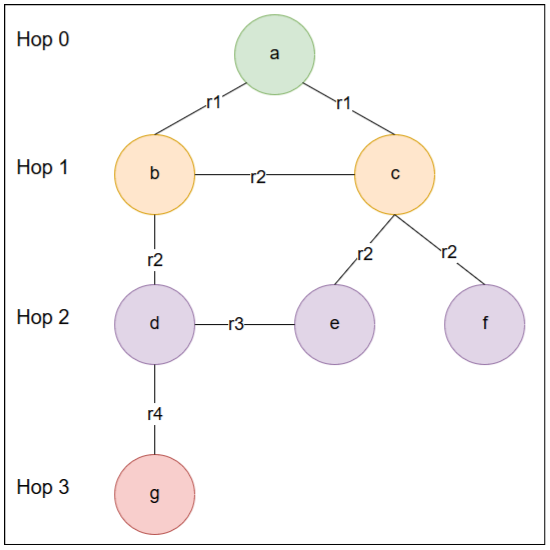
Now, we can calculate the weight for an edge label l in label vector of an entity e using information propagation factor of l.

$$Weight(e, l) = \sum_{i=1}^{n} {\alpha_{\mathrm{l}}}^i * N_{\mathrm{i}}(e, l) \tag{5}$$

Where, α_{l} is the information propagation constant for label l. $N_{\mathrm{i}}(e, l)$ represents number of occurrences of label l at exactly i neighborhood distance from entity e [1].

Algorithm 1 Label Vector Generation

```
1:  $E_T \longleftarrow$ Extract all entities with type T
2:  $\alpha \longleftarrow$ Information Propagation Constant
3:  $LabelVector \longleftarrow EmptyDictionary$
4:  for each $e \in E_T$ do
5:      $LabelVector[e] \longleftarrow \{e : 1\}$
6:      $visited \longleftarrow [e]$
7:      $previousLevelNodes \longleftarrow [e]$
8:      for $i \longleftarrow 1$ to $n$ do
9:          $currentLevelNodes \longleftarrow$ list of 1-hop distant nodes from previousLevelNodes
10:         for each $node$ in $currentLevelNodes$ do
11:             if $node$ not in $visited$ then
12:                 $LabelVector[e][node] \longleftarrow \alpha^i$
13:                 add node to visited list
14:             end if
15:         end for
16:         for each edge in edges at distance i from e do
17:             $LabelVector[e][edge] += {\alpha_{\mathrm{edge}}}^i$
18:         end for
19:     end for
20: end for
```



2-hop neighborhood node labels: a, b, c, d, e ,f
2-hop neighborhood edge labels: r1, r2

Label_Vector(a) =
{ a = 1,
b = 0.6,
c = 0.6,
d = 0.36,
e = 0.36,
f = 0.36,
r1 = 1,
r2 = 0.36
}

Consider entity a
If $\alpha = 0.6$

W(a, a) = 1
W(a, b) = 0.6
W(a, c) = 0.6
W(a, d) = $0.6^2 = 0.36$
W(a, e) = $0.6^2 = 0.36$
W(a, f) = $0.6^2 = 0.36$
W(a, g) = $0.6^3 = 0.216$

$\alpha_{r1} = \min(0.6, \frac{1}{2}) = 0.5$
$\alpha_{r2} = \min(0.6, \frac{1}{3}) = 0.33$
$\alpha_{r3} = \min(0.6, \frac{1}{1}) = 0.6$
$\alpha_{r4} = \min(0.6, \frac{1}{1}) = 0.6$

W(a, r1) = 0.5 * 2 = 1
W(a, r2) = $0.33^2 * 4 = 0.36$
W(a, r3) = $0.6^3 * 1 = 0.216$
W(a, r4) = $0.6^3 * 1 = 0.216$

Fig. 3. Label Vector Example

For the larger dataset, it is not possible to store all the label vectors in the main memory. We can store label vectors in a chunk of files. We can directly load these files serially to perform the further computation. This makes our algorithm scalable to the larger dataset.

5 Generating Top-K Representative Entities

Once we have a label vector for each entity of type T, we can generate top k representative entities using the k-means clustering algorithm. First, we choose random k samples from label vectors as our initial centroids for the clustering. We assign each entity from the entity list E_T to the nearest centroid by calculating cosine similarity scores. We apply a cosine similarity function on the label vector of an entity and the label vectors of centroids to calculate cosine scores. After the assignment step, we recalculate centroids by taking an average of all label vectors within the cluster. We repeat these steps until our centroids don't change. Also, we can limit our algorithm to perform only I iterations and return centroids. These centroids will be the representative entities of given type T.

5.1 Distance Function

We use cosine similarity between vectors to calculate the distance between the centroids and corresponding entities. Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. (1 - cosine score) is used as a distance measure between the label vector of the centroid and label vector of an entity. We use this distance to assign an entity to the nearest cluster in the K-Means clustering algorithm. Also, this distance is used to calculate the clustering cost.

The cosine similarity between two vectors is defined using the following equation:

$$CosineSim(\vec{v1}, \vec{v2}) = \frac{\vec{v1} \cdot \vec{v2}}{|\vec{v1}| \cdot |\vec{v2}|} \tag{6}$$

We can optimize our computations by normalizing label vectors before calculating cosine similarity. For each label vector, we calculate it's normalization factor (NF) by taking the square root of the sum of squares of each value in the label vector. We divide each value in a label vector by its normalization factor (NF) to get normalized label vector. Now, we can calculate the cosine similarity between two normalized label vectors by performing a dot product between them.

Algorithm 2 Label Vector Normalization

```
1: $v \leftarrow$ label vector of an entity
2: $NF \leftarrow 0$
3: for each $l$ in $v$ do
4:     $NF \mathrel{+}= v[l] * v[l]$
5: end for
6: $NF \leftarrow \sqrt{NF}$
7: for each $l$ in $v$ do
8:     $v[l] \leftarrow v[l]/NF$
9: end for
10: return $v$
```

Algorithm 3 Distance Function

```
1: $v1 \leftarrow$ Normalized label vector of an entity
2: $v2 \leftarrow$ Normalized label vector of an cluster centroid
3: $CommonLabels \leftarrow v1 \cap v2$
4: for each $l$ in $CommonLabels$ do
5:     $CosineScore \mathrel{+}= v1[l] * v2[l]$
6: end for
7: $distance \leftarrow 1 - CosineScore$
8: return $distance$
```

5.2 K-Means Clustring

K-means is the popular clustering algorithm which partitions a set of items into k clusters. Given a set of observations (x1, x2, ..., xn), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k sets (S1, S2, ... , Sk) so as to minimize the within-cluster sum of squares. Each cluster represents a set of entities which are closer or similar.

Once we express each entity of type T in multi-dimensional vectors and store it in files, we can apply k means clustering algorithm to partition them into k clusters. We can use the centroid of the resulting clusters as representative elements of entity type T.

In each iteration of K-means clustering, an entity is assigned to only one cluster whose centroid is the nearest to that entity. The cosine similarity is used to find the nearest centroid to the entity. After assigning every entity to the nearest cluster, we calculate new centroids by taking an average of all label vectors of entities in the same cluster. Generally, the ending condition for a k-means clustering algorithm is marked when the new centroids equal the previous centroids. However, we can also apply another stopping criterion by restricting the number of iterations of clustering. As the k-means clustering algorithm nearly converges after a few iterations, we can get pretty good results even after restricting it to have run for 6 to 8 iterations.

The main objective of the K-means algorithm is to minimize the clustering cost in each iteration by grouping the most similar entities in the same cluster. The clustering cost can be calculated using the following equation.

$$ClusteringCost = \sum_{i=1}^{k} \sum_{j=1}^{n_{\mathrm{i}}} distance(LabelVector[e_{\mathrm{j}}], LabelVector[c_{\mathrm{i}}]) \tag{7}$$

Where, k is number of clusters. n_{i} is number of entities in ith cluster. e_{j} is jth entity in ith cluster. c_{i} is a centroid of ith cluster.

We get k centroids after applying the k-means clustering. At each iteration of the k-means clustering, we calculate the new centroid of each cluster by taking an average of the label vectors of the entities within the same cluster. So, the resulting cluster centroids might not represent actual real-life entities. We can find the nearest real-life entity for that cluster by calculating distances between the centroid and all entities within that cluster. The entity which is nearest to the centroid is used as the representative entity for that cluster.

6 Optimization

Label vector generation is a memory intensive process. Each label vector is a multi-dimension representation of the single entity. If an entity is surrounded by many other entities and relations, then the label vector for that entity will have very large dimensions. For example, the average number of dimensions for the label vector of an entity of type 'Country' is around 100000. Also, there can be millions of entities of the single type in the larger dataset. For example, there are 232770 entities of type 'Film Character' present in the 'film' domain. In these cases, it is not possible to store the label vectors for each entity of given type in the main memory. One possible solution is to use files to store the generated label vectors. We define the count (N) for the maximum number of label vectors to be stored in the single file. In the label vector generation step, we dump every N number of the generated label vectors into a single file. After label vector generation we access these label vectors stored in files and apply k-means clustering algorithm on them. In this way, we solve the memory problem caused by larger datasets.

We need to calculate the distance between centroids and each entity in E_T for every iteration of the k-means clustering algorithm. There are N*I*K distance function calls we need to make for a single run of the k-means algorithm. Where N is the number of entities of the given type, K is a total number of required clusters and I is the total number of iterations in the k-means algorithm. These number of calculations are huge for a larger number of the entity set. It is very time consuming if we perform these operations serially. One possible solution to reduce the time required for computing these calculations is to use parallel processing. In our implementation, we have used python multiprocessing libraries for parallel computation of the distance function. Once we access a file of label vectors, we feed these label vectors to the pool of n processes. n is the user-defined integer number which indicates the number of parallel processes used. The variable n depends on the number of cores and the memory of an underlying hardware. The multiprocessing pool then calculates the distance between each entity label vector and all centroids. We use these distances to allocate an entity to its nearest cluster. The serial execution of the program for the 'Country' entity type took 572 minutes. The parallel solution with the pool of 4 processes completed in only 206 minutes to complete. Thereby, the parallel processing paradigm can be used to reduce the total time required for the program.

7 Experiments

We have used Freebase [3] datasets for our experiments. Freebase consists entity graphs from different domains such as 'film', 'location', 'book', 'people' , 'tv' and 'book'. We have performed experiments on the entity graphs of domain 'film', 'location' and 'tv'. Each entity graph is pre-processed. These datasets are stored and represented as RDF triples. Fig 4. represents the data distribution and total coverage of each domain in the freebase dataset.

The python program for generating representative elements fetch table data from the specified domain and store it in a required format. This processed data is used to generate label vectors for entities of the same type. By performing k-means clustering on label vectors, we get representative entities.

These experiments are performed on the Lonestar 5 server of the Texas Advanced Computing Center (TACC). The configuration of the Lonestar server is as follows: 24 cores, 64 GB RAM and 1TB secondary storage. This configuration is required only for the higher dataset and larger dimensional label vectors. Smaller dataset computations can be performed with much lesser hardware configuration.

For each experiment, label vectors of entities are generated with 2-hop neighborhood labels from that entity.

7.1 Experiment 1 (TV GENRE)

We have applied the algorithm to an entity type 'TV Genre' from the TV domain. There are total 306 entities of the type 'TV GENRE' present in the 'TV' domain. After generating label vectors, we applied k-means clustering to generate 10 representative entities. The average number of dimensions for each label vector of type 'TV GENRE' is 13619. Fig 5. represents the result of this experiment.

In the Fig. 5, we can clearly see that the related entities are grouped together in the same cluster. In the first cluster, the tv genres like 'Cooking Show', 'Cook-off', and 'Food' are closely related to the representative entity 'Cooking'. These entities are grouped together because they are mostly connected to the same tv episodes or tv series.

Representative entities for type 'TV GENRE' are as following: Cooking, New Year's Eve, Sports, Serial drama, Fashion, Games, Medical Fiction, News, Drama, Review, Sketch comedy, and Animation.

7.2 Experiment 2 (COUNTRY)

We have applied the algorithm to the entities of type 'Country' from the 'Location' domain dataset. There are total 590 entities of the type 'Country' present in the 'Location' domain. After generating label vectors, we applied k-means clustering to generate 15 representative entities. The average number of dimensions for each label vector of the type 'Country' is 931499. Fig 7. represents the result of this experiment.

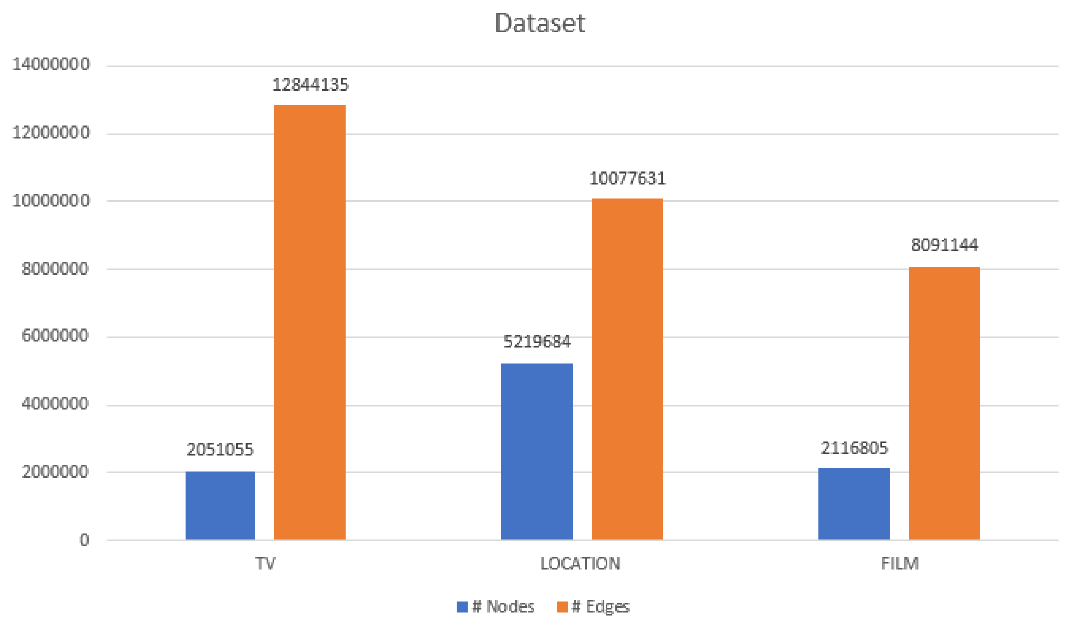


Fig. 4. Dataset Information

As a location dataset includes countries from historical data, most of the clusters are formed based on historical connections between different empires. Also, geographical closeness of the countries is also the factor in clustering most of the nearby locations in the same group.

Representative entities for type 'Country' are as following: the United States of America, Kingdom of Bulgaria, Faroe Islands, Francoist Spain, Portugal, Empire of Vietnam, Australia, Namaland, Zambia, Ivoire Coast, Albania, Svalbard and Jan Mayen.

7.3 Experiment 3 (Film Character)

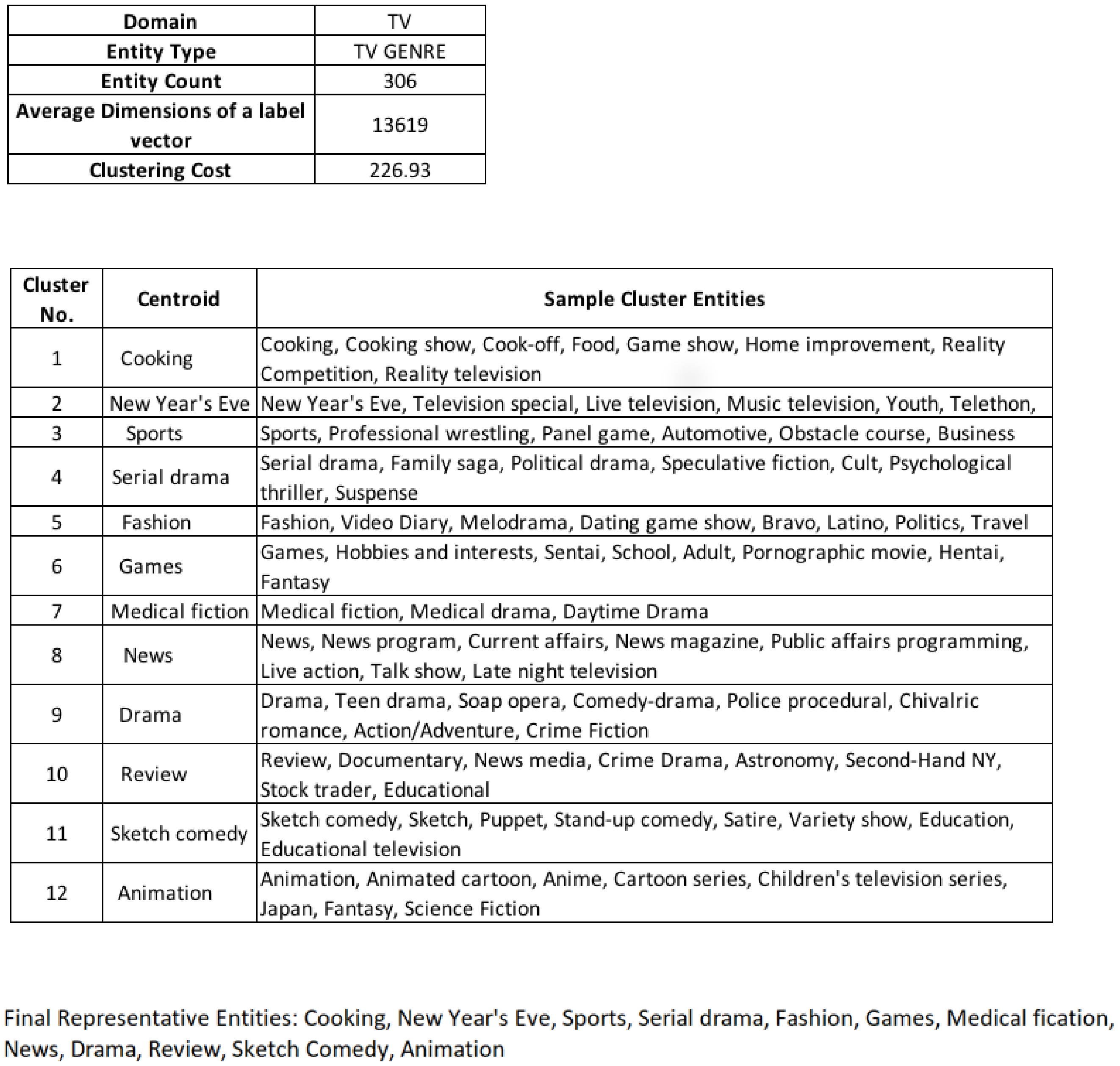
We have applied the algorithm to the entities of type 'Film Character' from the 'Film' domain dataset. There are 232770 entities of the type 'Film character' present in the dataset. We have generated label vectors for each entity of type 'Film Character'. Then we applied k-means clustering to generate 15 representative entities.

The resulting representative film character entities are diverse and well known. The film character dataset is huge and contains numerous variety of the film characters. So, some of the entities within a cluster may not be totally related to the centroid but they are closer to their cluster centroid than other centroids.

Representative entities for type 'Film Character' are as following: Immigration Cop, Jesus, Sherlock Holmes, Raj Malhotra, Superman, Abraham Van Helsing, Santa Claus, God, Captain Nemo, Esmeralda, Moe, Devil.

8 Conclusion

In this paper, we have described a method to generate representative elements of a particular entity type from the entity graph. It is a dynamic and scalable method which produces diverse and representative entities as a result of k-means clustering on label vectors. Based on the experiments, it can be very useful to generate preview tables mentioned in [2]. It will help the data worker to get a better understanding of the entity graph dataset

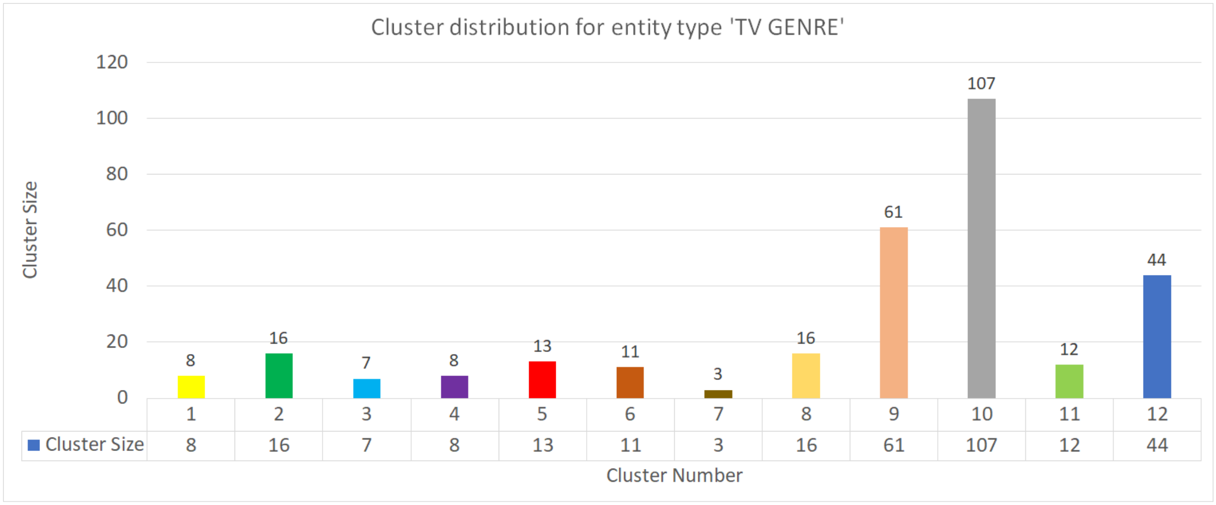


Domain	TV
Entity Type	TV GENRE
Entity Count	306
Average Dimensions of a label vector	13619
Clustering Cost	226.93

Cluster No.	**Centroid**	**Sample Cluster Entities**
1	Cooking	Cooking, Cooking show, Cook-off, Food, Game show, Home improvement, Reality Competition, Reality television
2	New Year's Eve	New Year's Eve, Television special, Live television, Music television, Youth, Telethon,
3	Sports	Sports, Professional wrestling, Panel game, Automotive, Obstacle course, Business
4	Serial drama	Serial drama, Family saga, Political drama, Speculative fiction, Cult, Psychological thriller, Suspense
5	Fashion	Fashion, Video Diary, Melodrama, Dating game show, Bravo, Latino, Politics, Travel
6	Games	Games, Hobbies and interests, Sentai, School, Adult, Pornographic movie, Hentai, Fantasy
7	Medical fiction	Medical fiction, Medical drama, Daytime Drama
8	News	News, News program, Current affairs, News magazine, Public affairs programming, Live action, Talk show, Late night television
9	Drama	Drama, Teen drama, Soap opera, Comedy-drama, Police procedural, Chivalric romance, Action/Adventure, Crime Fiction
10	Review	Review, Documentary, News media, Crime Drama, Astronomy, Second-Hand NY, Stock trader, Educational
11	Sketch comedy	Sketch comedy, Sketch, Puppet, Stand-up comedy, Satire, Variety show, Education, Educational television
12	Animation	Animation, Animated cartoon, Anime, Cartoon series, Children's television series, Japan, Fantasy, Science Fiction

Final Representative Entities: Cooking, New Year's Eve, Sports, Serial drama, Fashion, Games, Medical fication, News, Drama, Review, Sketch Comedy, Animation

Fig. 5. Result for entities of type 'TV GENRE'



	1	2	3	4	5	6	7	8	9	10	11	12
Cluster Size	8	16	7	8	13	11	3	16	61	107	12	44

Fig. 6. Cluster distribution of entities of type 'TV GENRE'

without downloading. Representative entities of a given entity type from an entity graph can be identified, which gives us an accurate preview of the dataset that could be used in real time applications.

Clustering helps to identify different data groups from the entities of the same type. The use of information propagation model and n-hop neighborhood entities proved vital to project. Each entity is represented in a multi-dimensional vector space. From the already stored graph data, similar entities were extracted, and the k means clustering was applied to divide entities into k different clusters of similar type entities.

The nearest entities to these cluster centroids were then grouped to be displayed as the representative entities in the preview table, thus projecting an educated view taking into consideration, the complete range of values.

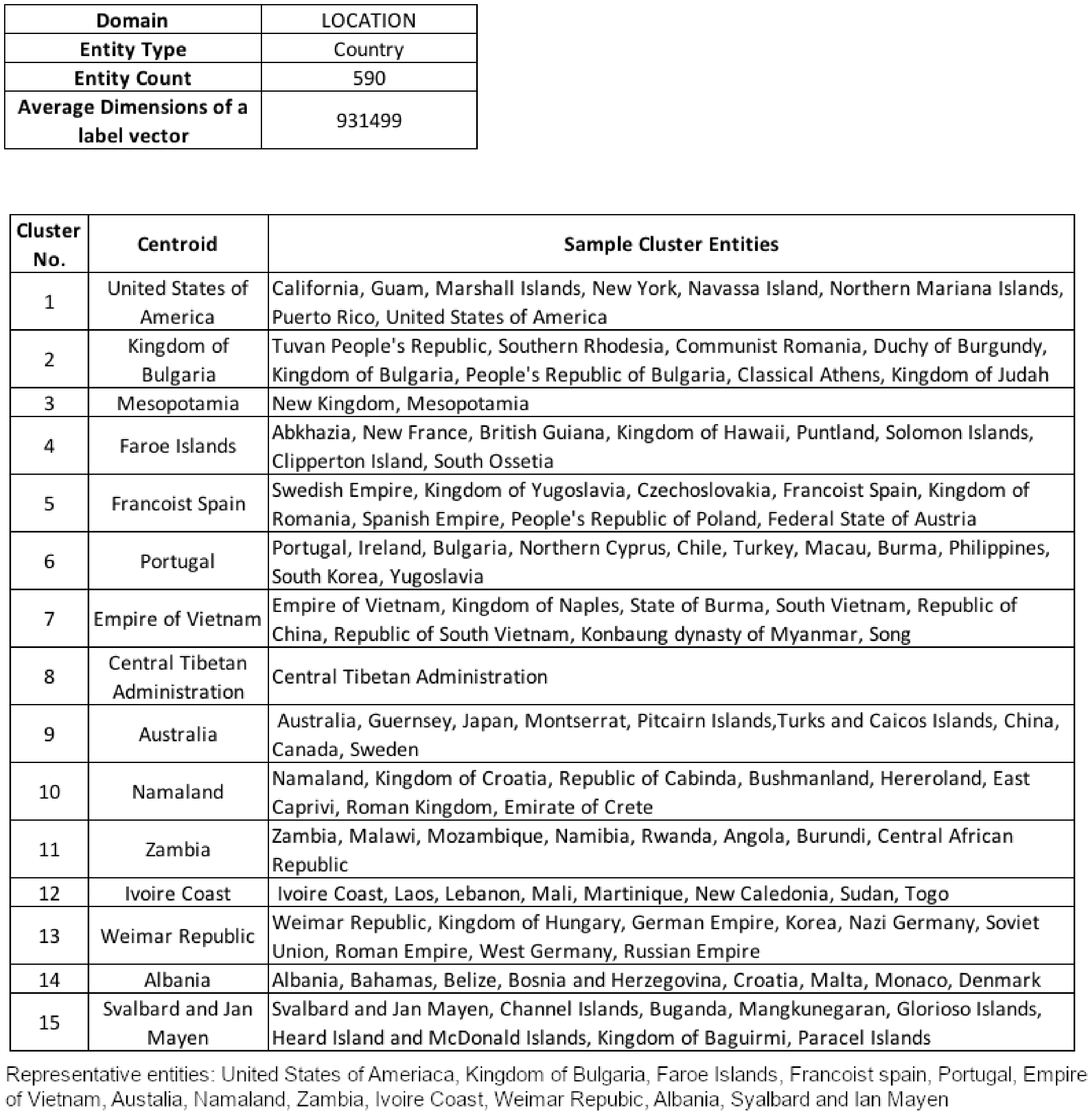
The memory challenge caused by large datasets is also solved by using files. In cases with large entities of similar type, it is unlikely to store label vectors for every other entity in the main memory. File storage is one prospect for a solution, which would store the generated label vectors. A part of the label vectors are stored in each file. These files are then accessed while applying the clustering algorithm.

The Freebase dataset is used for the experiments to find the representative entities for the preview table. The algorithm can be applied on a particular entity type from a dataset. Among a large entity set of similar type, label vectors would be generated for each entity type. Furthermore, k means clustering proves to be key in identifying the representative entities. The resulting representative entities are diverse and relatable. There are some instances where some entities within a cluster may not completely relate to the centroid but would be closer to their cluster centroid than others.

The method aids the selection of important samples from the entity graph. This, in turn, could be used in many applications that data science would cover.

9 Future Work

At a deeper introspection, there is room for improvement in a few areas of the algorithm. There are instances when the result of the experiment produces clusters with very few entities in it. This is because of the outliers in the dataset. We can improve the quality of the results by ignoring or removing outliers from the dataset.

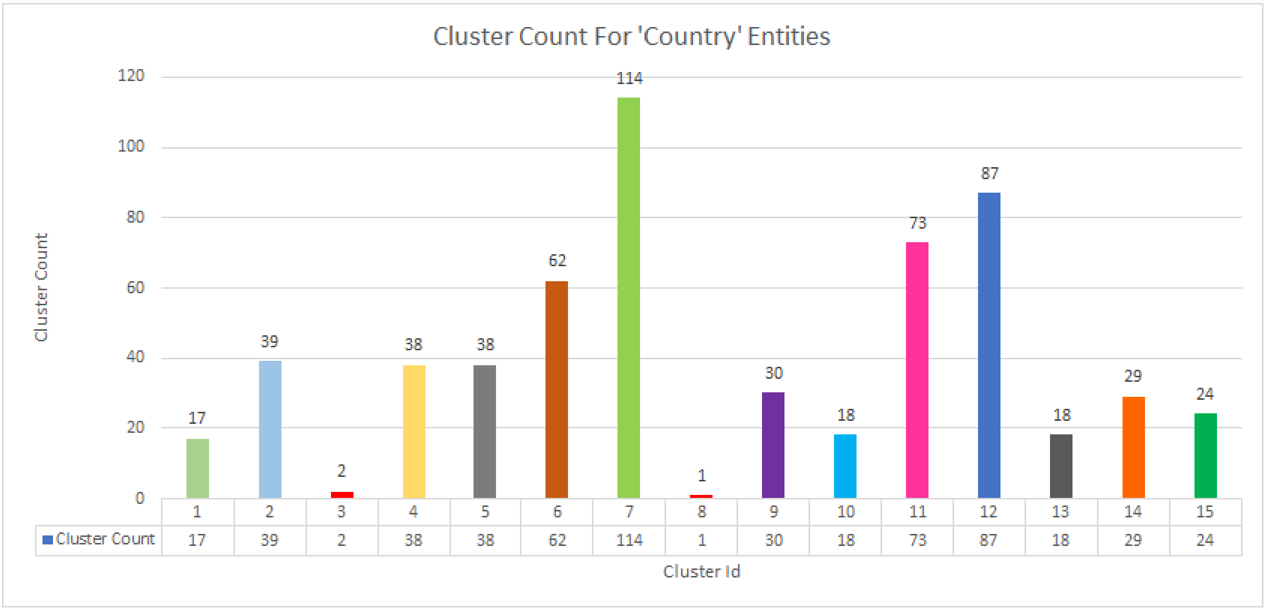


Domain	LOCATION
Entity Type	Country
Entity Count	590
Average Dimensions of a label vector	931499

Cluster No.	Centroid	Sample Cluster Entities
1	United States of America	California, Guam, Marshall Islands, New York, Navassa Island, Northern Mariana Islands, Puerto Rico, United States of America
2	Kingdom of Bulgaria	Tuvan People's Republic, Southern Rhodesia, Communist Romania, Duchy of Burgundy, Kingdom of Bulgaria, People's Republic of Bulgaria, Classical Athens, Kingdom of Judah
3	Mesopotamia	New Kingdom, Mesopotamia
4	Faroe Islands	Abkhazia, New France, British Guiana, Kingdom of Hawaii, Puntland, Solomon Islands, Clipperton Island, South Ossetia
5	Francoist Spain	Swedish Empire, Kingdom of Yugoslavia, Czechoslovakia, Francoist Spain, Kingdom of Romania, Spanish Empire, People's Republic of Poland, Federal State of Austria
6	Portugal	Portugal, Ireland, Bulgaria, Northern Cyprus, Chile, Turkey, Macau, Burma, Philippines, South Korea, Yugoslavia
7	Empire of Vietnam	Empire of Vietnam, Kingdom of Naples, State of Burma, South Vietnam, Republic of China, Republic of South Vietnam, Konbaung dynasty of Myanmar, Song
8	Central Tibetan Administration	Central Tibetan Administration
9	Australia	Australia, Guernsey, Japan, Montserrat, Pitcairn Islands,Turks and Caicos Islands, China, Canada, Sweden
10	Namaland	Namaland, Kingdom of Croatia, Republic of Cabinda, Bushmanland, Hereroland, East Caprivi, Roman Kingdom, Emirate of Crete
11	Zambia	Zambia, Malawi, Mozambique, Namibia, Rwanda, Angola, Burundi, Central African Republic
12	Ivoire Coast	Ivoire Coast, Laos, Lebanon, Mali, Martinique, New Caledonia, Sudan, Togo
13	Weimar Republic	Weimar Republic, Kingdom of Hungary, German Empire, Korea, Nazi Germany, Soviet Union, Roman Empire, West Germany, Russian Empire
14	Albania	Albania, Bahamas, Belize, Bosnia and Herzegovina, Croatia, Malta, Monaco, Denmark
15	Svalbard and Jan Mayen	Svalbard and Jan Mayen, Channel Islands, Buganda, Mangkunegaran, Glorioso Islands, Heard Island and McDonald Islands, Kingdom of Baguirmi, Paracel Islands

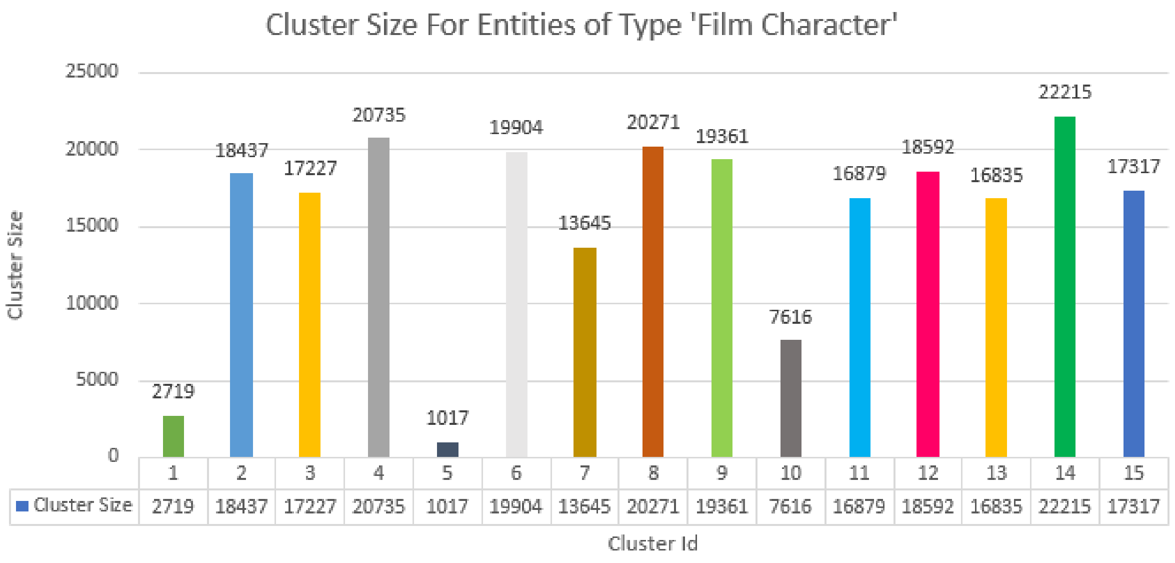
Representative entities: United States of Ameriaca, Kingdom of Bulgaria, Faroe Islands, Francoist spain, Portugal, Empire of Vietnam, Austalia, Namaland, Zambia, Ivoire Coast, Weimar Repubic, Albania, Syalbard and Ian Mayen

Fig. 7. Result for entities of type 'Country'



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
■ Cluster Count	17	39	2	38	38	62	114	1	30	18	73	87	18	29	24

Fig. 8. clustering distribution for entities of type 'Country'



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
■ Cluster Size	2719	18437	17227	20735	1017	19904	13645	20271	19361	7616	16879	18592	16835	22215	17317

Fig. 9. Cluster distribution of entities of type 'Film Character'

Domain	FILM
Entity Type	Film Character
Entity Count	232770
Average Dimensions of a label vector	13
Clustering Cost	210303.35

Cluster No.	Centroid	Sample Cluster Entities
1	Immigration Cop	Passenger to Airport, Pakistani Havaldar, Tourist at Airport, The Nurse, Man at the Park, Ship's Officer, Execution Squad Commander, Eric phatom
2	George Washington	Teresa, Bill Maplewood, George Logan, Pope, Mr. Maitland, Mrs. Destin, Dr. Astrid , Mohammed Jalaudin Ghaznavi
3	Jesus	Joey Crusel, Archbishop Gilday, Tristan, Theresa Wiggin, Leo Getz, Polly Watt, Roy O'Bannon, Anthony Gwynne
4	Sherlock Holmes	Arthur Holmwood, Allison Kerry, Gillian Guiler, Peter Venkman, Egon Spengler, Parker Selfridge, Colonel Miles Quaritch ,Benito Mussolini
5	Raj Malhotra	Rizwan Khan, Sumitra V. Patwardhan, Raj Aryan Malhotra, Sr. Inspector Shekhar Verma, Sher khan, Parimal Chaturvedi , Om Kapoor, Vikram Singh Rathore
6	Abraham Van Helsing	Lady Murasaki, J. Robert Fowler, Dagny Taggart, Ted "Theodore" Logan, Ming the Merciless, Benjamin Franklin, Thomas Jefferson , Alexander Hamilton
7	Santa Claus	Lieutenant Dan Taylor, Quasimodo, Irena Dubrovna, Kate Fitzgerald, Sarah Sanderson, Ted Carter,Henriette de France
8	Superman	Mister Fantastic, Clark Kent, Mrs Macready, Gawking Guy, Stephen Collins, Brother Roy, Lyndon B. Johnson, Inspector Devlin
9	God	Joseph, Attar, Maggie, Jefferey, Ángel Ferreyros
10	David Hall	Sailor Venus, Fozzie Bear, Tooth fairy, Kenny McCormick, Charlie Dog, Tasmanian
11	James Bond	Raoul, Gareth, Agent Stewart Dwight, FBI, Special Agent George Sledge, Agent Cooper, FBI Agent Winston, SBI Agent Megan O'Malley
12	Captain Nemo	Cohen the Barbarian,Yosemite Sam, Polonius, Mia Wallace, Edmond Honda, Jason
13	Esmeralda	Jasmine, Belle, Aurora, Princess Tam-Tam, Little Miss Amanda Martin,Little Mary, Moonyeen's Sister
14	Moe	Joker, Bean, Barbie, Charlie Brown, Sylvester, Rodney, Mr. Goodkat
15	Devil	Thug, Gangster, Scarecrow, Randolph Carter, The Spirit, Captain Sinclair, Roberta

Fig. 10. Result for entities of type ‘Film Character’

Currently, we are using random initialized starting centroids. Sometimes, random selection of the initial centroids may perform poorly as compared to the optimal solution. This problem can be resolved by using Kmeans++ algorithm mentioned in [6]] to choose initial centroids. There are a few problems with the underlying dataset.

Data normalization and refining of the data is necessary to get accurate results. For example, there are few entities like 'New York' which have type 'Country' in the location domain of the freebase dataset. These data problems can mislead the algorithm to produce unexpected results. It is quintessential to make sure that the data is robust to get accurate results. There is a direct ratio of the accuracy and efficiency of the preview tables to that of the semantic value, consistency, and integrity of the datasets used.

Another scope for future improvement can be the use of the distributed processing using Hadoop Map-reduce to further enhance the speed of processing. This can be beneficial when working on industry level time-intensive projects, where time priority pre-empts the need for cost efficiency.

Acknowledgments

I would like to thank Dr. Chengkai Li, Associate Chair of the Department of Computer Science and Engineering at the University of Texas at Arlington for his valuable guidance and expertise that greatly assisted the research. Also, I would like to thank Dr. Sona Hasani for sharing details of the information regarding preview tables. I am grateful to Dr. Bob Weems and Dr. Ramez Elmasri for their valuable comments which greatly improved the manuscript. Furthermore, I would like to thank Dr. Arijit Khan and his team for the idea of the Information Propagation Model which inspired this research work. My special thanks are extended towards my parents, family, and friends who have always supported me.

References

[1] Arijit Khan, Nan Li, Xifeng Yan, Ziyu Guan, Supriyo Chakraborty, Shu Tao: *Neighborhood based fast graph search in large networks.* SIGMOD Conference 2011: 901 to 912

[2] Ning Yan, Sona Hasani, Abolfazl Asudeh, Chengkai Li: *Generating Preview Tables for Entity Graphs.* SIGMOD 16, June 26 to July 01, 2016, San Francisco, CA, USA

[3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor: *Freebase: a collaboratively created graph database for structuring human knowledge.* In SIGMOD, pages 1247 1250, 2008.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives: *DBpedia: A nucleus for a Web of open data.* In ISWC pages 722 to 735, 2007

[5] F. M. Suchanek, G. Kasneci, and G. Weikum : *YAGO: a core of semantic knowledge unifying WordNet and Wikipedia.* In WWW, pages 697 to 706, 2007.

[6] Arthur D, Vassilvitskii S (2007) *'kmeans++: the advantages of careful seeding.'* Proceedings of the Eighteenth Annual ACM SIAM Symposium on Discrete Algorithms. pp 1027 to 1035