Bachelor Thesis

Answering approximated graph queries, embedding the queries and entities as boxes

Author: Teodor Aleksiev (2619071)

1st supervisor: dr. Michael Cochez
2nd reader: prof. dr. F.A.H. van Harmelen

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

August 26, 2020

Answering approximated graph queries, embedding the queries and entities as boxes

Teodor Aleksiev
VU Amsterdam
t.d.aleksiev@student.vu.nl

Abstract

Knowledge graphs are a suitable structure to efficiently store information in the form of entities and relations. Answering complex queries on such knowledge graphs, that can be incomplete, is a fundamental but yet challenging task. Recently Graph Convolutional Networks (GCNs) were used to encode the explicit information in the graph thus producing an embeddings, that serve various tasks (e.g. link prediction, entity classification, or query answering). Traditional methods embed entities and queries as points in the vector space, this can be problematic since even simple queries might have a large set of answers, and it is not clear how to encode a set with points. To counteract this problem we propose to embed the queries and the actual graph entities as boxes (i.e. axis-aligned hyper-rectangles), where a set of boxes intersecting the query box, correspond to a set of answers to that query. The model we are using consists of Relational-Graph Convolutional Network (R-GCN) (Schlichktrull et al. 2018) and relies on a specific scoring function, that takes into account size of intersection and the distance between the boxes.

1 Introduction

Data structures as graphs are extremely suitable for different applications, one of which is knowledge representation. They can store information in the sense of discrete entities from different domain linked to each other by relations of various types, thus resulting in a Knowledge Graph (KG). The relationships between the nodes are explicit, *e.g.*, *Alice* $\xrightarrow{worksAt}$ *VU*. Answering arbitrary logical queries over KG, like "*what are the project topics that both Alice and Bob work on?*", is a fundamental task in query answer retrieval, knowledge base retrieval, as well as more broadly AI (Ren et al. 2020). One of the common ways to retrieve an answer from a KG is to pose a structured query (for example, using the SPARQL query language (Harris et al. 2013)). In such cases the queries are answered via logical inferences based on the information present in the graph. However, usually the knowledge graphs do not contain all the information needed for accurate answer retrieval. Either because of their dynamic nature, when parts of the graph are constantly changed or updated, or because of the way the graph was constructed, some relevant information may be missing. Thus when trying to retrieve information from it there are cases that no answer is returned simply because of a missing relation to explicitly point to it.

Other approaches include representing the logical query as Directed Acyclic Graph and reason according to the DAGs structure to obtain a valid set of answers. This method has some drawbacks: (1) The direct subgraph matching becomes exponential in the query size, which leads to computational complexities, and cannot be scaled to the size of modern KGs: (2) Subgraph matching as well as structured queries are very sensitive to missing relations, and cannot answer queries that rely on them. To circumvent (2) one could impute missing relations (Nickel et al. 2015) or to reduce the constraint that must be met by an entity to be an answer (Fakou et al. 2017). But that would either make the graph denser, which would increase the computational complexity (1), or it might introduce wrong answers.

To circumvent these problems recent works propose an alternative approach. In it the queries as well as the KG entities are embedded into a low-dimensional vector space in a way that entities that are valid answers to a query are embedded close to the query (Hamilton et al. 2018; Guu et al. 2015). These approaches are robust in the task of link prediction and are order of magnitude faster, as the task answering a query is reduced to identifying the nearest entities to the embedding of the query in the vector space.

However embedding the queries and entities as vectors has its limitations. One cannot explicitly define a set of answers to a query if they are just embedded as point. To address this in our work we make use of a geometric embedding. We are embedding the entities and the queries as axis-aligned hyper-rectangles, or simply boxes in the vector space. Thus a query as a box, represent a closed region, which easily defines a set of answers when we define a valid answer to be an entity with a box that has some intersection with the query one. Thus we can easily extract multiple answers to queries. By embedding also the entities as boxes has the benefit that an entity box can overlap with multiple queries, even when the query themselves are not intersecting. Therefore an entity can take part as an answer to question of different context, for example *Alice* can be an answer to both queries regarding her *work* and her *family*, being part in both contexts as an *employee* and *family member* (See Fig. 1). Additionally the benefit of using *boxes* as an embedding is better than the use of *spheres*: (1) spheres grow symmetrically in all dimension, which makes them unsuitable for be-

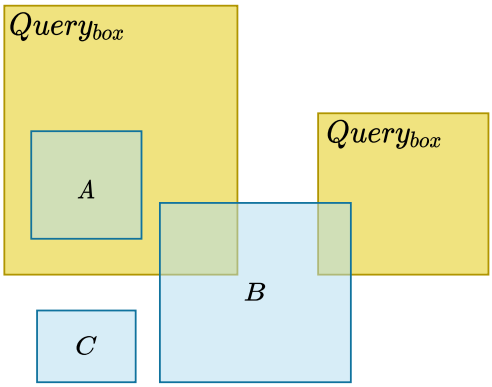

Figure 1: Example of a 2-dimensional box embeddings. Here there are two query boxes, and three entity boxes A,B, and C. In this case A is a valid answer to only one of the queries, B is embedded to be an answer to two non-overlapping queries of different contexts, and C is not an answer to any of the queries.

ing part in very different contexts, cause it will just make the entity or query sphere huge, while on the other hand, boxes (rectangles) can grow independently across the dimensions, (2) The intersection of two boxes is again a box, whereas intersection of spheres results in figure with a complex shape.

To address the problem of possible missing relations in the knowledge graph, we employ a Graph Convolutional Network (GNN). In particular we are using Relational-Graph Convolutional Network (R-GCN) (Schlichktrull et al. 2018), which takes into consideration the neighbours of the node and the relations that link them, thus improving on the task of link prediction, from which we benefit in our own task of query answering. The R-GCN performs a message passing on the graph of the query and then aggregates the result to produce the final box embedding. The method learns jointly the embeddings for the entities and the queries, which are trained and tested on three classical datasets, the knowledge graphs of which contain from thousands to million of entities and edges.

2 Related work

Multiple approaches for machine learning consider embedding the whole graph in vector space (Bordes et al. 2013; Wang et al. 2014; Yang et al. 2014). Their application is limited on the task of query answering. Regarding link prediction their methods need to consider all possible entities, which becomes exponential in the query size. More recent work (Daza and Cochez, 2020) proposes a message passing for query embedding (MPQE) method. Based on relational-graph convolutional network (R-GCN) (Schlichktrull et al. 2018) they directly encode the query into an embedding which is further optimized to be similar to the entity embeddings. This results in linear complexity in the size of the query. We closely use the architecture of the MPQE model. They encode the query and entities as vectors, for which calculate the cosine similarity score. The drawback is that there is no clearly defined threshold after which an entity is considered not a valid answer. In contrast to their method, our resulting vector embeddings for the query and the entities is interpreted as a box in the vector space, thus we use the box borders to serve as a natural threshold to disregards the valid from non-valid answers.

Second line on related work is regarding structured embeddings which associated knowledge base concepts with geometric objects as regions (Vilnis et al. 2018; Ren et al. 2020). In (Vilnis et al. 2018) they develop probabilistic model for lattices based on hypercube embeddings that can model both positive and negative correlations. Their model embeddings and similar probabilistic query embeddings (POE) of Lai (Lai and Hockenmaier 2017) represent subsets of probabilistic event space which are directly integrated. Embedding the entities as hyper-cubes (or boxes) they are able to retrieve multiple answers to a given query, but yet again as in MPQE, a clear threshold cannot be defined.

In contrast Query2Box (Ren et al. 2020) embeds the actual queries as axis-aligned hyper-rectangles (boxes) and entities as points in lower dimensional vector space. Same as them we use the geometric property of the box to be enclosed, thus defining a natural border to be used when classifying if a certain entity is a valid answer. Additionally we also embed the actual entities as boxes to address possible different contexts a specific entity might take part in. Even though geometric objects were used to model individual entities and pairwise relations between them in previous works (Vilnis et al. 2018; Vendrov et al. 2015) we are using the geometric objects to model sets of entities and reason over them. In this sense our work is also related to the classical Venn Diagrams (Venn 1880), where the boxes are essentially the Venn Diagrams in the vector space, and the presence or lack of intersection between them is used as an indicator of answer correctness.

3 Problem definition

We use a similar definition as in (Daza and Cochez, 2020). A Knowledge Graph (KG) is defined as a tuple $(\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{T})$, where $\mathcal{V}$ is a set of nodes representing entities, and $\mathcal{E}$ a set of typed edges connecting the nodes. A function $\tau : \mathcal{V} \rightarrow \mathcal{T}$ assigns a type to every node, where $\mathcal{T}$ is a set of entity types. Each edge in $\mathcal{E}$ corresponds to a relation between two nodes v_i and $v_j \in \mathcal{V}$, that we denote by $r(v_i, v_j)$, where $r \in \mathcal{R}$ is a relation type.

Given the KG, we can ask queries that seek certain entities based on some conditions. A way to define them is to use conjunctive queries, which are a subclass of Existential Positive First-order (EPFO) logic. They consists of *conjunctions* of binary predicates, where entities (as anchors) and query variables serve as arguments.

To better illustrate this, consider a KG of an academic institution, where students work on specific topics, and these topics are related to their projects. Thus we can pose the following query:"get all the cities C, such that university U is located in C, and both *Alice* and *Bob* lectured at U." The answers for that query are all the entities P which satisfy the

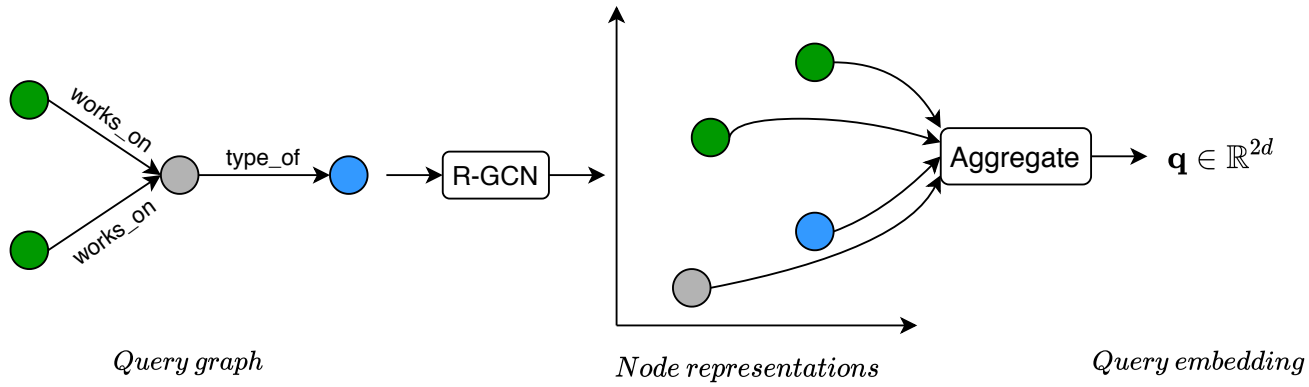

Figure 2: The architecture of the model we are using (MPQE) (Daza and Cochez, 2020) takes as input a query graph and outputs the final query embedding. The features of the graph nodes are embedding of entities in the KG. An R-GCN propagates information across the graph, and an aggregation function is used to retrieve the final query embedding in $\mathbb{R}^{2d}$.

the following condition:

$$C.\exists U, C: \text{ located\_in}(C, T) \wedge \text{lectured\_at}(\textit{Alice}, U) \wedge \text{lectured\_at}(\textit{Bob}, U). \quad (1)$$

These conjunctive queries are formally defined as follows:

$$q[V_?] = V_? \,.\, \exists V_1, \ldots, V_k : e_1 \wedge e_2 \wedge \ldots \wedge e_n, \quad (2)$$

where $e_i = r(v_a, V), V \in \{V_?, V_1, \ldots, V_k\}, v_a \in \mathcal{V}, r \in \mathcal{R}$ or $e_i = r(V, V'), V, V' \in \{V_?, V_1, \ldots, V_k\}, V \neq V', r \in \mathcal{R}$. In the notation v_a represents a non-variable anchor entity, $V_1, \ldots, V_k$ are existentially bound variables, and $V_?$ is the target variable (answer) of the query. Thus an entity is considered an answer to the posed query if it satisfies the defined conditions. The goal is to find the set of entities that are valid answers and satisfy the query even when some of the binary predicates would require a missing edge from the KG. To address that problem and efficiently model a set of entities in the vector space we use *boxes (i.e. axis-aligned hyper-rectangles*. The benefits of this method is that unlike single points, a box has an interior, thus if an entity embedded as a box is and answer to a query also embedded as a box, we model the entity box to have an intersection with the query one. We operate on $\mathbb{R}^d$. Every entity $v \in V$ has an assigned embedding $\mathbf{e}_v \in \mathbb{R}^{2d}$. In addition an embedding method for the query is defined that maps the full query to a vector $\mathbf{q} \in \mathbb{R}^{2d}$. We further define a box in $\mathbb{R}^d$ by $p = (Cen(p), Off(p)) \in \mathbb{R}^{2d}$ as:

$$Box_p = \{v \in \mathbb{R}^d : Cen(p) - Off(p) \preceq v \preceq Cen(p) + Off(p)\}, \quad (3)$$

3.1 Scoring

We define two variants of scoring function for each entity in the KG, that will be used in the learning process. Since box embedding is described as:

$$\mathbf{p} = (Cen(p), Off(p)) \in \mathbb{R}^{2d},$$

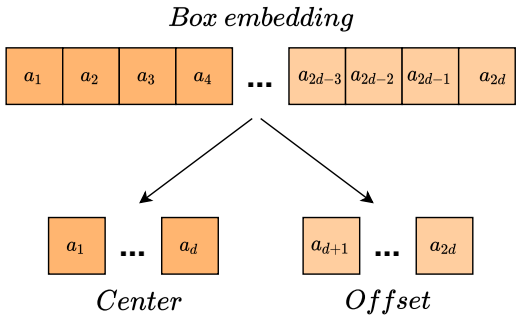

Figure 3: Visualization of the Box embedding vector ($Box(p) \in \mathbb{R}^{2d}$), in which the left half represents the Center vector of the box ($Cen(p) \in \mathbb{R}^d$), and the right half is the Offset vector ($Off(p) \in \mathbb{R}^d$).

The dimension-wise length of each side of a single box is determined as follows:

$$l_{\mathbf{p}} = 2 * Off(p) \in \mathbb{R}^d,$$

we also retrieve the dimension-wise overlap of the respective box sides:

$$l_{\mathbf{inter}} = Max(\mathbf{0}, Min(\mathbf{p}_{max}, \mathbf{q}_{max}) - Max(\mathbf{p}_{min}, \mathbf{q}_{min})),$$

where Min and Max are element-wise functions, $\mathbf{p}$ and $\mathbf{q}$ are two boxes for which the intersection is calculated, and the min and max subscripts are defined for the both boxes as follows: $\mathbf{p}_{max} = \text{Cen}\{\mathbf{q}\} + \text{Off}\{\mathbf{q}\} \in \mathbb{R}^d$, $\mathbf{p}_{min} = \text{Cen}\{\mathbf{q}\} - \text{Off}\{\mathbf{q}\} \in \mathbb{R}^d$.

Actual Volumes In the case of partial (full) overlap between two boxes (across all dimensions) the resulting figure in the embedding space is again a box. Then by having the length of the sides the actual box (or the intersection box) the volume of it is:

$$Volume = \prod_{i=0}^{n} l_i$$

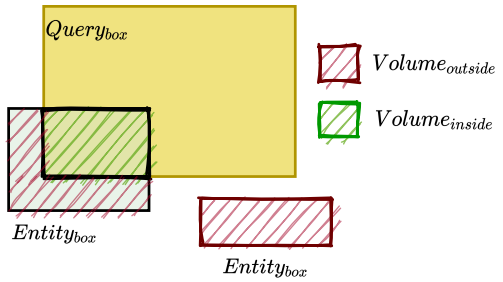

Figure 4: 2-dimensional sketch of entity boxes and query box, where one of the entities has an overlap with the query box, again of shape of a box.

where n is the total dimensions, l_i is the side length of the box in dimension i. In case of calculating the volume of the intersection box l_i represents the length of the overlap in the corresponding dimension. To retrieve the volume of the entity box which is outside the query one (Fig. 4), we simply subtract the intersection volume ($Volume_{inside}$) from the total box volume ($Volume_{total}$) since both of them are axis-aligned hyper rectangles, but the shape of the outer part is more complex:

$$Volume_{outside} = Volume_{total} - Volume_{inside} \quad (4)$$

The expected drawback of that metric has to do with two extreme cases caused but the high dimensional embedding space. The first is that boxes with side lengths across, a lot of dimensions, below 1 will result in an a very small $Volume$, with the risk to be rounded zero. The second is the opposite the volume can become exponentially large of the sides are too wide. Additionally in the case where at least in one dimension there is no overlap at all the resulting $Volume_{inside}$ will be zero until and intersection is achieved across all dimensions.

Pseudo Volume To address the aforementioned probable problems with the calculation of the real box volume, we propose a custom function that yet again make use of the dimension-wise intersection of the sides. We denote it as $PseudoVolume$ and calculate it as follows:

$$PseudoVolume = \sum_{i=0}^{n} ln(1 + l_i) \quad (5)$$

where n is the total dimensions, l_i is the side length of the box in dimension i. In case of calculating the volume of the intersection box l_i represents the length of the overlap in the corresponding dimension. We are taking the normalized logarithm to account for the case of no intersection, which will lead to a side of zero and $-inf$ as a result in the summation. The benefit of that function is whenever there is an intersection in at least one dimension the $PseudoVolume_{inside}$ value is not zero, thus the model has a changing value to help the training. In order to obtain the $PseudoVolume_{outside}$ for each dimension we take the difference between the full side length of the box and the intersection length.

Distance Because initially there might not be any intersection between the query and its targets, and this will pose setback in the loss function, since the intersection volume will be zero until by chance they intersect, we define an additional metric to help the learning process and to decrease time of convergence by introducing the distance from the **entity box** center to the closest point on the query box. This distance is defined as:

$$dist_{outside}(\text{Cen}\{\mathbf{e}\}; \mathbf{q}) = \quad (6)$$
$$\|Max(\text{Cen}\{\mathbf{e}\} - \mathbf{q_{max}}, \mathbf{0}) + Max(\mathbf{q_{min}} - \text{Cen}\{\mathbf{e}\}, \mathbf{0})\|_1$$

where $\mathbf{q}_{\max} = \text{Cen}\{\mathbf{q}\} + \text{Off}\{\mathbf{q}\} \in \mathbb{R}^d$, $\mathbf{q}_{\min} = \text{Cen}\{\mathbf{q}\} - \text{Cen}\{\mathbf{q}\} \in \mathbb{R}^d$ (Fig. 5).

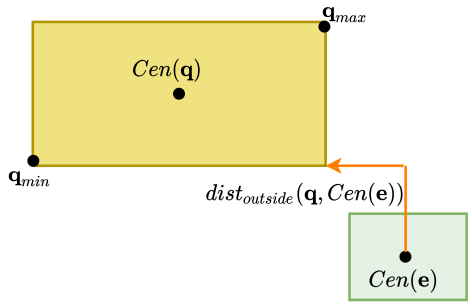

Figure 5: The Manhatan distance (L1 norm) between the center of the entity box and the nearest corner of the query box.

Score The final score has two forms depending on the choice of *Volume* calculation method. By using the actual volumes of boxes in the embedding space, the resulting score function used during learning is:

$$score_{volume}(\mathbf{e}, \mathbf{q}) = -ln(1 + Volume_{inside}) \quad (7)$$
$$+ ln(1 + Volume_{outside}) + dist_{outside}(\text{Cen}\{\mathbf{e}\}, \mathbf{q})$$

where $\mathbf{e}$ is the entity box, $\mathbf{q}$ is the query box, $Volume_{inside}$ is the volume of the intersection box between the query and entity boxes, $Volume_{outside}$ is the volume of the part of the entity box that is not intersecting with the query box, and $dist_{outside}$ is the distance in the vector space between the entity box center and the nearest point part of the query box. We are taking the logarithm of the volume to scale down its value to be comparable to the distance metric, and use one plus the volume to account the case where there is no intersection and $Volume_{inside} = 0$. For the case of $PseudoVolume$ the score we obtain is defined as follows:

$$score_{pseudovolume}(\mathbf{e}, \mathbf{q}) = - PseudoVolume_{inside} \quad (8)$$
$$+ PseudoVolume_{outside},$$

where $PseudoVolume_{inside}$ is the value for the intersection box, calculated dimension-wise, and $PseudoVolume_{outside}$ is the value for the part of the entity box that is not overlapping with the query.

4 Model definition

At first we need to set the initial embedding for the entities in the graph and the variable types. Since we are dealing with boxes and each embedding is a concatenated vector containing the $Cen(e) \in \mathbb{R}^d$ and the $Off(e) \in \mathbb{R}^d$, we need to take this into account, and not to begin with a fully random embeddings, which will be boxes all over the embedding space with various sizes. To address this problem we propose to sample the Center vector from a uniform distribution, this results in a more evenly distributed boxes in the embedding space.

$$Cen(e) \sim \mathbb{U}(a, b) \tag{9}$$

To solve the problem of having random size boxes, we sample the Offset vector from a normal distribution, to produce relatively similar box sizes.

$$Off(e) \sim \mathbb{N}(\mu, \sigma^2) \tag{10}$$

Then we follow a similar procedure to (Daza and Cochez, 2020). When initializing the nodes on the query graph, each one (v) has a feature vector, which is given by a one-hot representation $h_v^{(0)}$ with a total of N elements, where N is the amount of entities in the graph. In the case of v being a variable node in the query graph and not a known anchor it is represented by $\mathbf{h}_v^{(0)}$ with $|\mathcal{T}|$ elements, where $\mathcal{T}$ is the amount of different node types. This one-hot embedding serves as a key to a look-up matrix with the initial box embeddings.

We define a matrix of *entity embeddings* $\mathbf{M}_e \in \mathbb{R}^{2d \times N}$, where $2d$ is the dimension of the embedding space, because each box is $(Cen(p), Off(p)) \in \mathbb{R}^{2d}$, and *type embeddings* with the corresponding matrix $\mathbf{M}_t \in \mathbb{R}^{2d \times \mathcal{T}}$. The resulting node embedding function is as follows:

too much about concrete implementation

$$\mathbf{h}_v^{(1)} = emb(\mathbf{h}_v^{(0)}) = \begin{cases} \mathbf{M}_e \mathbf{h}_v^{(0)} & \text{if} v \in \mathcal{V}_{qe} \\ \mathbf{M}_t \mathbf{h}_{v_t}^{(0)} & \text{if} v \in \mathcal{V}_{qv} \end{cases} \tag{11}$$

The result is that each entity has its own embedding as a box in the vector space, and each variable in the query graph is initialized by a representation of its type.

After we have defined node features for the query graph, we proceed of applying L steps of *message passing* with a GNN. In order to address the influence from the neighbour nodes and the task of link prediction, we employ a Relational Graph Convolutional Network (R-GCN) (Schlichktrull et al. 2018). The benefits of the model are taking into account the type of the relations involved and the neighbour nodes in the graph, when updating the features of a node. We are using the standard R-GCN propagation rule where at step $l+1$ the representation of node v is given as follows:

$$\mathbf{h}_v^{(l+1)} = f\left(\mathbf{W}_0^{(l)} \mathbf{h}_v^{(l)} + \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_v^r} \frac{1}{|\mathcal{N}_v^r|} \mathbf{W}_r^{(l)} \mathbf{h}_j^{(l)}\right), \tag{12}$$

where $h_j^{(l)}$ is the hidden state at layer l for node j, $h_v^{(l)}$ is the hidden state at layer l for node v, f is non-linearity (in our case $ReLU$), $\mathcal{N}_v^r$ is the set of indices of neighbours of node v through relation type r, and $\mathbf{W}_r^{(l)}$ is a relation specific weight matrix,and $\mathbf{W}_0^{(l)}$ is the weight matrix for self-connections in the R-GCN.

After L applications of the R-GCN layer, the representations of all the nodes from the query graph can be combined to produce the vector in $\mathbb{R}^{2d}$ that acts as a box embedding of the query itself, by means of *aggregation* function ϕ:

$$\mathbf{q}_\phi = \phi\left(\{\mathbf{h}_v^{(L)} | v \in \mathcal{V}_{qe} \cup \mathcal{V}_{qv}\}\right); \tag{13}$$

we continue with the definition of few options for that function.

Traditional aggregation functions can leverage the embedding representations of other nodes in the query graph. Simple permutation invariant functions include the sum and maximum of:

$$\mathbf{q}_{SUM} = \sum_{v \in \mathcal{V}_q} \mathbf{h}_v^{(L)} \quad , \quad \mathbf{q}_{MAX} = max(\mathbf{h}_v^{(L)}, v \in \mathcal{V}_q) \tag{14}$$

Given the method we interpret the query embedding as a box, with the final goal of answer entities to be embedded with an overlap to that box, using the $\mathbf{q}_{SUM}$ will result in larger box offsets and a further center location.

For alternative aggregation function we make use of Target Message (TM) aggregation function (Daza and Cochez, 2020). Let us denote with D the *diameter* of the query, to represent the longest shortest path between two nodes in the graph. In the query structures we are considering (Fig. 3), $d \in [1, 3]$. By noting that at most D steps are needed to propagate messages from all the query nodes to the target node, an adaptive query embedding method is used. Given the specific query structure and its diameter D the method performs D steps of message passing in the R-GCN, and then it selects the representation of the target node:

$$\mathbf{q}_{TM} = \mathbf{h}_{T_q}^{(\mathcal{D})} \tag{15}$$

We also consider an aggregation function that considers additional parameters (Hamilton et al. 2017), it passes all the final representations of nodes taking part in the query through a Multi-Layer Perceptron (MLP) and then sums the result, which is uses as embedding for the query:

$$\mathbf{q}_{MLP} = \sum_{v \in \mathcal{V}_q} \text{MLP}\left(\mathbf{h}_v^{(L)}\right) \tag{16}$$

The parameters of the model consist of entity and type embeddings and the parameters of the R-GCN itself. As in previous work on the task of query embedding (Ren et al. 2020; Hamilton et al. 2018; Mai et al. 2019; Daza and Cochez, 2020) we optimize the model using gradient descent. Given a training set of queries q and their embedding $\mathbf{q}$, we optimize a negative sampling loss (Mikolov et al. 2013):

$$\begin{aligned} L = &-log\, \sigma(\gamma - score(\mathbf{e}^+, \mathbf{q})) \\ &- \sum_{i=1}^{k} \frac{1}{k} log\, \sigma(score(\mathbf{e}^-, \mathbf{q}) - \gamma), \end{aligned} \tag{17}$$

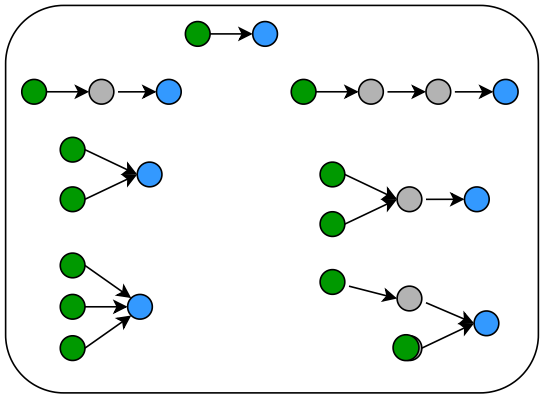

Figure 6: Used query structures for evaluation on query answering. **Green** nodes correspond to anchor entities, **gray** nodes are the variables in the query, and the **blue** nodes represent the targets (answers) of the query.

where γ represents a fixed scalar margin, $\mathbf{e}^+$ is a positive sample that represents an entity in the knowledge graph that answers the query, and $\mathbf{e}^-$ are negative samples which are entities taken at random that are not a valid answer, but has the same type as the target.

5 Experiments

We evaluate the performance of the model on the task of query answering over incomplete knowledge graphs, by considering 7 different query structures (see Fig. 6). These structures are chosen such that to be comparable to other related work (Ren et al. 2020; Daza and Cochez, 2020; Hamilton et al. 2018; Mai et al. 2019). Namely we use three types of chain queries with the range from one to three connecting relations (1-, 2-, and 3-chain), two intersection structures that are composed of different amount of anchor edges (2- and 3-inter), and two complex ones that combine chain and intersection parts (3-chain-inter and 3-inter-chain). The goal is to test the model on the additional task of retrieving multiple valid answers to a query, thus we also train and test on queries with more than one answer, whereas in the validation test phase these queries rely on missing edges in the KG.

5.1 Datasets

For our experiments we use publicly available knowledge graphs that have been used in other literature of graph representation learning (Schlichktrull et al. 2018) and query answering (Daza and Cochez, 2020; Hamilton et al. 2018) containing from thousands to millions of entities:

- **AIFB**: a KG of an academic institution, where entities are persons, organizations, projects, publications, and topics;
- **MUTAG**: a KG of carcinogenic molecules, where entities are atoms, bonds, compounds, and structures;
- **AM**: contains relations between different artifacts in the Amsterdam Museum, including locations, makers, and others;

Table 1: Statistics of the knowledge graphs that were used for training and evaluation.

	AIFB	MUTAG	AM
Entities	2,601	22,372	372,584
Entity types	6	4	5
Relations	39,436	81,332	1,193,402
Relation types	49	8	19

Table 2: Average number of multiple answers to different queries structures, across the used datasets.

	AIFB		MUTAG		AM	
Structure	Train	Test	Train	Test	Train	Test
1-chain	3.4	1.2	1.9	1.1	1.2	1.0
2-chain	34.5	6.4	13.4	4.7	10.2	3.5
3-chain	**47.0**	**7.2**	**17.6**	**5.4**	13.8	3.7
2-inter	9.3	3.2	1.6	1.3	9.1	3.5
3-inter	5.1	2.8	1.0	1.0	7.4	2.9
3-inter-chain	15.5	4.2	1.9	1.7	10.3	3.5
3-chain-inter	22.8	5.6	2.6	2.3	**15.2**	**4.4**

A table with their statistics can be found in table 1.

5.2 Query generation

To obtain query graphs, we sample sub-graphs from the KG, following the structures shown in Figure 2. For each query depending on the structure, we sample anchor nodes and also the relations between them, the variable nodes and the target nodes. Then for that set of anchors, relations and structure we retrieve all the targets that are answers to the specific query. In the case of *2-chain* and *3-chain* queries in some cases the amount of valid answer is too high (over 100 000), this happens due to existence of more than one central nodes connected to multiple others via the same relation used in the query. This is observed mainly in the datasets of AIFB and MUTAG, whereas in the case of AM we observe similar behaviour in the query structure *3-chain-intersection*, but yet again high values for the cases of *2-chain* and *3-chain*. thus we introduce a threshold of 100 answers for a given query, in order to train on multiple different queries, but to keep up to 100 valid answers per query. See table 2 for more details on the the average number of targets after query generation. Additionally for each query we obtain a negative sample, these are entities that are invalid answers. For the case of intersection queries, also a hard negative sample is obtained, an entity that would be a valid answer of the conjunction is relaxed to a disjunction.

5.3 Evaluation

The performance of the method is judged by queries that require information not present in the graph during the training

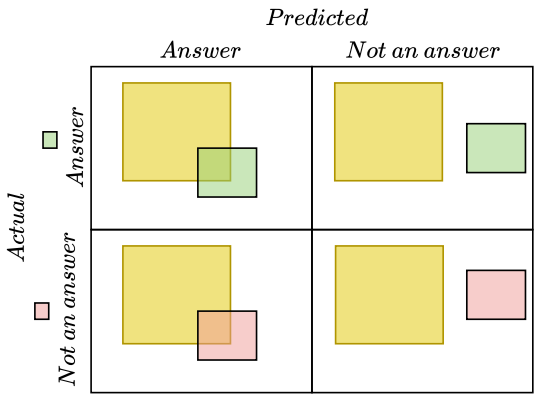

Figure 7: Model of the confusion matrix used for evaluation of the results, the yellow box is representation of a query, the green box and the red box are respectively a valid and a invalid answer to the query.

phase. To achieve that initially we marked 10% of the graph edges with label *removed*. During the query generation step for each set of anchors, relations and a target we check if all the edges are present in the graph and not marked as *removed*, if so this query will be part of the training set, these queries are used to optimize eq. (17). In the case where at least on of the query edges is *removed* with probability of 10% we store it in the validation set, otherwise it is part of the test set. In such a way we ensure that queries used for validation/test contain at least one unseen edge in their graph. This is done in order to evaluate the method performance beyond the traditional graph traversal techniques, which would not find an answers because of the missing edges.
From the incomplete graph 2 million query targets are extracted and each respective query has between 1 and 100 valid answers. Additionally 300 000 query targets and their corresponding queries are used for testing, and 30 000 are used for validation in order to perform early stopping during the training phase, the results are reported on the test set.
For the final evaluation we are using the box embeddings of the query and the entities (the valid answers and the negative ones) to retrieve a binary score. The score reflects on whether there is an intersection between the query box and the entity box, meaning it is embedded as a valid answer, and whether there is no overlap, thus not being an answer. Then with the obtained binary scores a confusion matrix is constructed to give the precision and the recall of the method on the specific query type (Fig. 7).

5.4 Model

We evaluate the performance of the model under the different aggregation functions (*SUM*, *MAX*, *TM*, *MLP*). All initial embeddings we initialize using the stated protocol of sampling the *Center* form an *uniform* distribution in the range $[0, 10]$, and the *Offset* from a *normal* distribution with $mean = 3$ and $std = 1$. We are using 3 R-GCN layers, which is in correspondence with the *TargetMessage* aggregation function, in which the amount of message passing steps is given by the query diameter (in our case the maximum diameter of a query is 3). In the case of the Multi-Layer Perceptron aggregation (MLP) we use similarly to (Daza and Cochez, 2020) two fully-connected layers. In all the cases the nonlinear function used in the models is ReLU. We minimize eq. 17 using the Adam optimizer with a learning rate of 0.01, using an embedding of 64 dimension, which means the effective boxes are embedded in a 32 dimensional vector space. For the implementation PyTorch and the PyTorch Geometric library were used (Fey and Lenssen 2019). As a baseline we consider the Graph Query Embedding (GQE) method by Hamilton et al. (Hamilton et al. 2018) and also the Message Passing Query Embedding (MPQE) method by Daza and Cochez (Daza and Cochez, 2020). The main differences between the two are the use of a projection and intersection operators in the case of (Hamilton et al. 2018) and the use of R-GCN layers in (Daza and Cochez, 2020). Since both show comparable results tested on the data sets considered in the current work - AIFB, MUTAG and AM (Daza and Cochez, 2020), but MPQE does not always outperform GQE we will use the both.

6 Results

The model was trained for over 200 000 iterations, but still we suspect it has not reached a convergence. We observe that still no actual intersection occurred between the boxes of the queries and the target nodes. We believe that it has to be trained for more time, and maybe even to try lowering the amount of train/test queries. In order to still produce some results we examined the trained models on the AIFB dataset, each with a different aggregation function. Since currently we cannot evaluate it based on the presence of intersection in order to produce a proper confusion matrix, we use the L1 distance metric we defined in eq. (7) to check whether at least the target entity box is "moving" closer to the query embedding in comparison to a negative node of the same type. Even though the model using the Multi-Layer Perceptron was trained for a reasonable amount of iterations, it still shows no improvement. Thus a further extensive testing on the parameters used will be performed. On the other hand both agregations *SUM* and *TM* show reasonable and close results. *TM* has slightly better performance which we explain with the fact that it takes into account the diameter of the query and uses the representation of the target node from the last layer, whereas the *SUM* just sums the embedding of all node taking part in the query graph. We observe a notably higher results on queries that contain chain structures in comparison to the intersection ones, which further motivates an experiment that trains the model only on 1-chain queries.

7 Future experiments

Regarding future testing on the model we are planning thorough analysis on its performance using the MUTAG and the AM datasets. Both have greater amount of entities differing in magnitude, also an increase in the number of the relations in the graph in comparisson to AIFB (see Table 2). Regarding the types of entities and relations in the three datasets,

Table 3: Percentage (%) of answers embedded closer to the query box compared to a non answer, with regard to the query structure, using different aggregation function. Tested on AIFB dataset.

	AIFB		
Structure	SUM	TM	MLP
1-chain	67.48	**69.84**	0.0
2-chain	68.78	**75.85**	0.0
3-chain	76.55	**79.86**	0.0
2-inter	62.09	**63.10**	0.0
3-inter	63.32	**63.35**	0.0
3-inter-chain	67.61	**67.91**	0.0
3-chain-inter	68.87	**72.43**	0.0

all of them have relatively same entity types, whereas the main difference is observed in the relation types. Thus we will perform an extensive test if any context clusters are formed in the final embedding space. With MUTAG having only 8 types of relations we are expecting the entities that share context to be grouped closer to each other in the vector space, and to observe lesser amount of and more condensed groups (or clusters) in comparrisson to the other two datasets (AIFB with 49 relational types, and AM with 19). We will check if for more uniform knowledge graphs, like MUTAG, that hold information for strictly specific area (e.g. carcinogenic molecules) it is easier to define and observe context. MUTAG is observed to have reasonably low average amount of multiple answers in more complex queries that include intersection (see Table 1.

We would perform test against all three datasets, by training only on 1-chain queries, and testing on all seven query structures. As stated by (Daza and Cochez, 2020) this line of training also proves to be successful in the task of query answering even on structures that were not explicitly trained. We will check that claim against our box embedding model and the respective loss function.

A further investigation on the resulting size of the box embeddings in the vector space will be conducted. The goal is to check whether query boxes enlarge or shrink in accordance to the amount of valid answers they contain, thus we will answer the question if the query box size is relevant of the amount of answers. Additionally to check if the entities remain embedded as boxes with relatively large size of they tend to shrink to an extent they can be just embedded as point, thus we will make a direct comparison to the Query2Box (Ren et al. 2020) method, where indeed they embed the entities as points in the vector space. Thus we will gather insight if the use of boxes as embedding for the entities is actually more beneficial in the task of multiple answer retrieval to incomplete knowledge graph, or the results are comparable to the case where entities are points.

Regarding the scoring function we plan an extensive test on which variant gives better results. Thus the $score_{volume}$ and $score_{p}seudovolume$ will be compared not only based on the final results in embedding, but also the time it takes for the model to converge, since $Volume_{inside}$ changes from zero only when there is an intersection across all dimensions, whereas $PseudoVolume_{inside}$ has impact on the score already when at there is an intersection in at least one dimension.

Additionally an extensive hyper-parameter testing. We will test different ranges for initial embedding regarding the input parameters to the *uniform* and the *normal* distribution parameters used in eq. (9) and eq. (10). Testing on different values of the scalar margin γ used in the loss function, and different number of the R-GCN layers needed (beside the case of *TM* aggregation function, where the layers depend on the size of the query).

8 Conclusion

In this work we presented a new method for query answering, that combines and further develops the use of embedding both queries and entities in a lower vector space, as axis-aligned hyper-rectangles, and the use or a Relational-Graph Convolutional Network to perform the message passing between the relevant nodes, taking into account their neighbours and the relations connecting them. We defined a custom scoring that depend on the presence of intersection between the box embedding (either across all dimensions or at least on one) and the distance between the boxes. Due to constraints during training we were able to test it partially only on one of the planned Knowledge Graph, but still proves promising in the results. Further extensive experiments are needed to conclude and adequately compare the performance of the model with other models dealing with the task of query answering.

References

Daza, Daniel; Cochez, Michael 2020. Message Passing for Query Answering over Knowledge Graphs *arXiv preprint arXiv:2002.02406*.

Schlichtkrull, Michael; Kipf, Thomas N; Bloem, Peter; Van Den Berg, Rianne; Titov, Ivan; Welling, Max 2018. Modeling relational data with graph convolutional networks In *European Semantic Web Conference (pp. 593-607). Springer, Cham.*.

Hamilton, Will; Bajaj, Payal; Zitnik, Marinka; Jurafsky, Dan; Leskovec, Jure 2018. Embedding logical queries on knowledge graphs In *Advances in neural information processing systems (pp 2026–2037)*.

Bordes, Antoine; Usunier, Nicolas; Garcia-Duran, Alberto; Weston, Jason; Yakhnenko, Oksana 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems (pp 2787–2795)*.

Wang, Zhen; Zhang, Jianwen; Feng, Jianlin; Chen, Zheng 2014. Knowledge graph embedding by translating on hyperplanes. In *Aaai* (Vol. 14, No. 2014, pp. 1112-1119).

Yang, Bishan; Yih, Wen-tau; He, Xiaodong; Gao, Jianfeng; Deng, Li 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

Vilnis, Luke; Li, Xiang; Murty, Shikhar; McCallum, Andrew 2018. Probabilistic embedding of knowledge graphs with box lattice measures. *arXiv preprint arXiv:1805.06627*.

Ren, Hongyu; Hu, Weihua; Leskovec, Jure 2020. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. *arXiv preprint arXiv:2002.05969*.

Lai, Alice; Hockenmaier, Julia 2017. Learning to predict denotational probabilities for modeling entailment. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers (pp 721–730)*.

Mai, Gengchen; Janowicz, Krzysztof; Yan, Bo; Zhu, Rui; Cai, Ling; Lao, Ni 2019. Contextual graph attention for answering logical queries over incomplete knowledge graphs In *Proceedings of the 10th International Conference on Knowledge Capture (pp 171–178)*.

Mikolov, Tomas; Chen, Kai; Corrado, Greg; Dean, Jeffrey 2013. Efficient estimation of word representations in vector space In *arXiv preprint arXiv:1301.3781*.

Vendrov, Ivan; Kiros, Ryan; Fidler, Sanja; Urtasun, Raquel 2015. Order-embeddings of images and language In *arXiv preprint arXiv:1511.06361*.

Venn, John 1880. I. On the diagrammatic and mechanical representation of propositions and reasonings In *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 10(59):1-18.

Harris, Steve; Seaborne, Andy; Prud'hommeaux, Eric 2013. SPARQL 1.1 query language In *W3C recommendation*, 21(10), 778.

Nickel, Maximilian; Murphy, Kevin; Tresp, Volker; Gabrilovich, Evgeniy 2015. A review of relational machine learning for knowledge graphs In *Proceedings of the IEEE*, 104(1), 11-33

Fokou, Géraud; Jean, Stéphane; Hadjali, Allel; Baron, Mickael 2017. Handling failing RDF queries: from diagnosis to relaxation In *Knowledge and Information Systems*, 50(1), 167-195.

Guu, Kelvin; Miller, John; Liang, Percy 2015. Traversing knowledge graphs in vector space In *arXiv preprint arXiv:1506.01094*.

Hamilton, Will; Ying, Zhitao; Leskovec, Jure 2017. Inductive representation learning on large graphs In *Advances in neural information processing systems*, (pp. 1024-1034).

Fey, Matthias; Lenssen, Jan Eric 2019. Fast graph representation learning with PyTorch Geometric In *arXiv preprint arXiv:1903.02428*.

@articlefey2019fast, title=Fast graph representation learning with PyTorch Geometric, author=Fey, Matthias and Lenssen, Jan Eric, journal=arXiv preprint arXiv:1903.02428, year=2019